CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE

Canada

NAME OF AUTHOR/*NOM DE L'AUTEUR* Shane Don Caplin

TITLE OF THESIS/*TITRE DE LA THÈSE* TEACHING OF A COMPUTER PROGRAMMING LANGUAGE BY A SELF-DIRECTED COURSE OF STUDY

UNIVERSITY/*UNIVERSITÉ* Simon Fraser University

DEGREE FOR WHICH THESIS WAS PRESENTED/*GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE* Master of Arts (Education)

YEAR THIS DEGREE CONFERRED/*ANNÉE D'OBTENTION DE CE GRADE* 1983

NAME OF SUPERVISOR/*NOM DU DIRECTEUR DE THÈSE* Dr. A. J. (Sandy) Dawson

DATED/*DATÉ* March 21, 1983 SIGNED/*SIGNÉ* [redacted]

PERMANENT ADDRESS/*RÉSIDENCE FIXE* [redacted]

TEACHING OF A COMPUTER PROGAMMING LANGUAGE BY A SELF-DIRECTED COURSE OF STUDY

by

Shane D. Caplin

B.Sc. Simon Fraser University 1979

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ARTS (EDUCATION)

in the Faculty

of

Education

SIMON FRASER UNIVERSITY

March 1983

APPROVAL

Name: Shane Don Caplin

Degree: Master of Arts (Education)

Title of Thesis: Teaching of a Computer Programming Language by a Self-Directed Course of Study

Examining Committee

Chairperson: L. Prock

A. J. Dawson
Senior Supervisor

R. J. D. Jones
Assistant Professor

W. Muir
Associate Professor
Faculty of Education
University of Victoria
External Examiner

Date approved Mar. 21, 1983

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

TEACHING OF A COMPUTER PROGRAMMING LANGUAGE BY A SELF-DIRECTED COURSE OF STUDY

Author: [redacted]
(signature)

Shane Don Caplin
(name)

March 21, 1983
(date)

ABSTRACT

A method is proposed for teaching computer programming by means of a self-directed study course. The course is developed using a combination of textbook, study guide and taped instructions. The thesis presents a theoretical background for such a course. A study guide was developed to supplement textbook instruction. Nine audio tape lectures provide audio contact between the student and the instructor.

As the development of the course in this thesis is a pilot project only, suggestions are made for the way in which further work can be done in self-directed methods of teaching computer programming. Comments are also made about similar courses for students who are interested in computer science as a tool rather than as a discipline, and for those students who wish to teach computing in the junior and senior secondary schools.

TABLE OF CONTENTS

LIST OF TABLES

A. Introduction

The current interest in the use of computers in secondary - and in some instances elementary - schools suggests that we are in the midst of a fast growing technological movement. Increasing numbers of people in all disciplines are finding it necessary to learn computer programming. The growth has been so rapid that there has been little time to consider how best to implement and how best to teach this new technology.

With the rapid emergence of the use of computers in our society and the inclusion of computer instruction in the educational curriculum, there has come about a need to establish some relationship between the theoretical and the applied aspects of computer programming and to develop an effective method for the teaching of this discipline.

Many of the current educational methods tend to lack a firm philosophical or pedagogical basis. The field of teaching of computer programming is so new that the published research is relatively sparse.

One of the specific problems is that until quite recently there were almost as many programming styles as there were programming languages. However, the research carried out by E. W. Dijkstra (1965, BIT 1968, ACM 1968, 1970) has shown that the development times for large systems can be significantly reduced by the use of "structured programming".

Early programmers, by necessity, were recruited from the scientific disciplines. These professionals learned how to use a computer either by reading programming manuals from the computer manufacturers or by learning how to use a computer directly from a manufacturer's representative. As highly trained professionals, these people looked upon programming as something that could be learned in one's spare time using the traditional methods that were successful in the learning and teaching of science. This attitude still exists today and for this reason the method of teaching programming in many learning institutions is to give the students little more than a description of the features of a particular programming language and tell the students to "get on with it". Up to now, they were not taught structured programming.

The fact that computers are being used in ever increasing numbers has created a need for more teachers of computer programming. However, in a survey conducted in 1977, Gopaulsingh (1977) pointed out that the schools in British Columbia have not been able to meet this growing need. The shortage of computing science teachers is even more acute today.

One effective alternative to class teaching which can utilize structured programming is self-directed study (Allen, 1980). Materials are developed to assist the student in grasping concepts and implementing these concepts in the form of practical exercises, thus diminishing the need for direct instruction and supervision.

One of the greatest problems in introducing computer programming into the classroom is that at present, most teachers do not have the expertise to deal with the subject (Gopaulsingh, 1977). Moreover some teachers feel threatened by a new discipline they do not understand.

Even when a single area in computer science such as programming is singled out as the basis for a computer course, the design and implementation of the course can be very complex, because certain fundamental questions must be answered, such as: should the computer be in control of the student or should the student be in full control of the computer? This touches on the problem of the way in which rigidity would be built into the instructional program if the computer were in control compared with the decision-making by the student in the more flexible student-control situation. The latter approach is the one which is emphasized in this thesis.

It seems that a good self-directed package in computer programming should include the following features:

a. the theory of learning

b. the theory of computer programming

c. practice and experience, i.e., the immediate application of the principles learned.

d. distributed practice

The aim of this thesis is to develop an approach to teaching individuals to become competent computer programmers, using methods by which students can learn relatively independently of a classroom instructor. To be acceptable to the university and to the students, such a self-directed study course must have a sound theoretical basis in both the theory of learning and the theory of computer programming.

This thesis will outline a beginning programming course that is the result of the research of educators working in the area of learning theory and computer science, as well as of the current research of the author.

B. Survey of the Literature

A survey of the research in computer science education and teaching methods shows that there is a scarcity of pertinent material in this field. There is considerable subsidiary information such as correspondence courses and audiovisual material for people working in areas other than the learning of computer science. Most of such material is in the form of computer assisted instruction (CAI) and computer managed instruction (CMI). Material dealing with the computer as the subject of instruction rather than as the mechanism of instruction is sparse.

I. Individualized Instruction

It should be pointed out here that the ultimate goal of this thesis is to devise a method by which students may study and learn computer programming independently of an instructor.

While there is a substantial amount of literature on individualized instruction, it is the area of independent study and its subset, self-directed study, that relates most directly to this thesis.

> "Individualized instruction is not the same as independent study. To be sure, independent study is often a part of individualized instruction. But individualized instruction is the larger idea. It includes independent study." (Esbensen, 1968).

Independent study can itself be described as consisting of two major elements: individual study - study by oneself, and self-directed study - study independent from a regularly structured curricula (Gibbons, 1971).

In a review conducted on individualized instruction by Barbara and Marcel Goldschmid (Goldschmid and Goldschmid, 1972), the concepts of the audio-tutorial approach and modular instruction were shown to be successful teaching methods.

In the audio-tutorial approach audio-tapes are designed to direct the student in various types of learning activities. It was pointed out that this method works well when supplemented with laboratory work, readings, slides and films (Goldschmid, and Goldschmid, 1972).

Modular instruction, another form of individualized instruction, combines many of the advantages of other instructional innovations such as performance ojectives, self-pacing, and frequent feedback. Modules consist of self-contained units of planned learning activities. These activities may be in the form of reading textbooks and articles, of listening to audio-tapes, examining diagrams, etc. The advantages of modular instruction are self-pacing, being able to frequently identify student strengths and weaknesses, and the ability to easily provide extra help for weak students through the design of remedial modules (Goldschmid and Goldschmid, 1972).

Self-instructional materials are an important part of a self-directed study course. The types of work material chosen depend largely on how far this material is seen to be replacing the function of the teacher. The most structured pattern of material is that of programmed learning where the student can progress in small steps and each step is tested immediately. In this way the student can, with little or no outside assistance, keep track of how well he is learning (Davies, 1980).

II. Self-Pacing/Self-Directed Study

The problem that our learning institutions are faced with is not one of providing more individuals to teach courses, but rather of providing those teachers with a better background and teaching aids so that the frustrations mentioned by Gopaulsingh (1977) in the following quotation can be minimized.

> "As Lawler (1970) pointed out, in many situations innovations have been introduced into the schools without careful analysis of the appropriateness and sophistication of the content and materials for the student group. Often not enough attention has been given to relationships with other courses and to the level of development of the individual students. Frequently, in-service teacher preparation has been neither appropriate nor adequate. The result has been that subsequent to the innovations, frustration, resentment and feelings of inadequacy have surfaced."

A self-directed study course in computer programming prepared by professionals could be very helpful in overcoming the shortage of trained instructors of computer programming. In addition to making sure that the students are taught the essentials of the discipline, each student has the advantage of individualized

instruction. Students proceed at their own speed and at their individual competency level. A student who is already familiar with a certain aspect of computer progamming may skip over those particular instructional topics. No time is wasted on needless review of material already learned (Allen, 1980).

A course in introductory Fortran programming using a set of TV lectures was set up on an experimental basis at Texas A & M University (Simmons, Ward and Thompson, 1974). Although this course was not designed as a self-directed study course, the main conclusion of the experiment was that both students and instructors were motivated toward designing the length and frequency of class meetings to meet with their particular needs. It was verified that when computer programming was taught in this manner a wide variety of teaching programs could be implemented without adversely affecting student learning or student interest.

The Keller method of instruction (Keller, 1968) is characterized by self-directed study with a high degree of student help and discussion.

> "Self-pacing allows students to spend a variable amount of time on basic skill-building. As a result, well-prepared students can move ahead of the class and poorly prepared students can take the time to 'gear up' before attempting difficult problems. ... Emphasis on student self-help seems to be a good way to build self-confidence." (Young, 1974)

The Keller system of teaching (Keller, 1968) emphasizes:

1. the go-at-your-own-pace feature which allows the student to plan a personal schedule;

2. the use of lectures and demonstrations as vehicles of motivation rather than sources of information;
3. the use of tutorial assistants to make frequent assessments and lend a personal aspect to the educational process.

III. Audio-Tutorial Method

In a treatise on the audio-tutorial system of instruction, James Russell describes the advantages of the use of this system:

> "Individual students respond differently to various learning activities, media, and rates of learning. The audio-tutorial system provides a multi-faceted, multi-media approach to learning which is under the control of the student. It utilizes an audio tape to tutor the student through the instructional activities and media until he had mastered the object of the lesson...
>
> During a typical session, the student is actively involved in learning at his own pace. He listens, reads, writes, manipulates learning materials, makes observations, and performs experiments." (Russell, 1978)

In looking at the question of achievement, Russell quotes the studies conducted by Fisher and MacWhinney (1976) which found the audio-tutorial system to be far superior to conventional methods of teaching.

The audio-tutorial method developed by Stephen Lower at Simon Fraser University has been very sucessful in the teaching of undergraduate chemistry courses (Lower, 1981). This method combines audio tapes with printed material to achieve a more efficient presentation than either medium could accomplish when

used separately. According to Lawer the audio-tutorial offers:

1. self-pacing: students can stop and replay the tape wherever and whenever they wish;
2. interaction of visual and aural inputs which may have significant effects on learning and retention;
3. the spoken language with its elements of pace, intonation and pauses.

An interesting study was done on how graduate student performance and attitudes were affected by the use of audio tapes as a medium for learning (Barringer and Bekiroglu, 1978). This new approach was introduced in order to help overcome the difficulties in mastering the material of a very intensive operations research course that was being offered to MBA students. In this study two groups in two different geographic locations were selected. In order not to prejudice the results the same syllabus and examinations were used by both groups.

GMAT (Graduate Mangement Admission Test) scores were used to ensure that the two student groups were drawn from the same population. The mid-term and final examination scores of the course provided the measures of performance for group comparisons.

The format of this new approach to teaching the course involved supplementing a set of audio tapes with corresponding condensed printed discussions, or lecture notes, which students could follow on a step-by-step basis to cover the essentials of the course. In this way the students were able to improve their

preparation prior to class meetings.

The experiment in audio taped lectures involved using two groups of students. One group used the audio tapes for the first half of the course while the second group used the tapes for the second half of the course. The same examinations were administered to both groups in order to eliminate variations due to different examinations and different subject material when analyzing student performance with and without tapes.

Conclusions of this study were as follows:

1. The audio tapes seemed to be more useful to those students who were struggling with the material than those who were able to master the material easily. However, both types were shown to be helped by the tapes.
2. The tapes seemed to improve exam scores for those students who used them and the improvement seemed directly proportional to the amount of use.
3. The students showed a very favourable attitude toward the availability of the tapes and recommended them to other students.
4. Barringer and Bekiroglu feel that these results can be obtained in any course by using learning modules with audio tapes. The factors that seem to be at work here are:
   a. Audio tapes seem to encourage students to spend more time in preparing for topics and offer an alternative to the student who is "bogged down" in the text book.

b. The "alternative approach" was usually related to the way the instructor viewed the subject and thus helped the student to emphasize the points which the instructor felt were important.

c. Students appreciate an instructor who makes a special effort to improve the quality of the course.

IV. Concept Development

As computer science has developed, a certain number of ideas have come to be recognized in the field of programming (Papert, 1979). Among these ideas is the concept of the modularity of pure procedures and the concept of "top-down, structured programming".

> "We have found it incorrect to assume that beginning programming students are unable to handle 'higher level' concepts. We have found it both reasonable and worthwhile to present the topics of algorithm development, stepwise refinement, recursion, and topdown modular programming along with details of a particular language." (Schneider, Weingart and Perlman, 1978)

Current education practices involve using 'manipulatives', such as coins, sticks or blocks to make procedures more concrete. In the field of computer programming languages, DuBoulay and his colleagues (DuBoulay & O'Shea, 1976; DuBoulay, O'Shea & Monk, 1980) used 'manipulatives' to provide a concrete model for teaching LOGO to children. The results of tests and experiments conducted at the University of California on this model, and other methods of teaching, show that there is clear and consistent evidence that a concrete model can have a strong

effect on the encoding and use of new technical information by novices (Mayer, 1981).

In an earlier paper Mayer asks what knowledge is acquired by a novice learning BASIC programming, and how can this knowledge be more efficiently acquired. In answering these questions, Mayer suggests that students will be able to learn more easily and more efficiently if they develop three basic skills that are not obvious either in instruction or in traditional performance, namely:

1. the ability to analyze each programming statement into a type of pre-statement. A pre-statement is a set of "transactions" corresponding to a line of code;
2. the ability to enumerate the transactions involved for each pre-statment (A transaction is an event that occurs in the computer and involves some operation at some storage location.);
3. the ability to "chunk" prestatements into general clusters or configurations.

The paper shows through the results of extended psychological testing of students how the use of this method of teaching enhances student performance (Mayer 1979).

It would seem from Mayer's results that by simply teaching the syntax of a programming language and the way to use the syntax, ultimately produces very poor programmers. In other words, it seems that the level of student performance is directly related to how much a student understands about the

programming language, and how much the student understands about the computer itself.

Structured programming is a term that occurs innumerable times in the software literature. Although the exact meaning of what structured programming is varies from author to author, a typical definition is: "Structured programming is a manner of organizing and coding programs that makes the programs easily understood and modified" (Donaldson, 1973). It is generally agreed that unstructured programming is inefficient and prone to errors simply because unstructured design is unpredictable. One goal then is to remove the unpredictable component from programming. In other words, if software is to become more reliable, we should reduce the complexity of the programs that are being written. One way of achieving this is to reduce large problems to smaller, more easily manageable sub-problems. Some of the benefits of this reduction in complexity are fewer testing problems, increased programmer productivity, and improved program clarity, maintainability, and modifiability (Jensen, 1981).

Most educators agree that to be most effective, training should include theory, demonstration, practice, feedback, and classroom application. Learning outcomes can be classified as: awareness, the acquisition of concepts or organized knowledge, the learning of principles and skills, and the ability to apply those principles and skills in problem solving activities. These classifications apply equally well to the teaching of

computer programming.

If someone attempting computer programming for the first time has already acquired a systematic, well-disciplined approach to problem analysis, learning to program a computer just means learning the syntax of the programming language to be used. However, few students have this prerequisite training in problem solving and it is this training and knowledge that is more important than learning the nature and syntax of a programming language. Whether a student in a programming course is learning programming as part of another course, or is progressing independently, there are several points that can be identified as "what not to do". Some teachers, usually science teachers, tend to take a "grammatical" approach, the logical step-by-step exposition of the grammar of the computer language. This way of teaching differs from a pedagogical approach by which students first learn to use the language quickly (Bork 1981).

> "In the logical approach, students take a long time to get to the point where they can write programs, and they do not gain much feel for the art of programming, because they spend so much of their time on the rules of grammar" (Bork, 1981)

Bork points out that at one time this strategy was widely used in the teaching of foreign languages but has since been discredited.

Bork also points out it is a fallacy to believe that a student must learn all about a programming language and must learn the full language. It important only for a student to

absorb the principles of programming and it is not necessary to be completely familiar with all the facilities that a programming language has to offer. As a matter of fact it has been demonstrated that a small subset of a particular programming language is usually more than sufficient for the beginning student's needs.

V. Specified Objectives

An experiment at the Univerity of Delaware using videotaped instruction showed results similar to the experimental course at Texas A & M (Simmons, Ward and Thompson, 1974). Here the emphasis was on opening and closing each session with the behavioral objectives of the lesson. Every statement or concept definition was supported with pertinent examples and reinforced with built-in quizzes (Hartman and Caroly, 1971).

Although there are many methods documented in the literature for the design and development of instructional materials (Langdon, 1978; Davies, 1980; Goldschmid and Goldschmid, 1972; Russell, 1978; and Drumheller, 1971), the "Systems Approach Model" (Dick and Carey, 1978) seemed to relate best to the author's view on how to approach the design of a self-directed study course in computer programming.

The "Systems Approach Model" uses a complete structured algorithmic type of approach to the design of a curriculum. The model involves attacking the problem of design and

implementation by following a set of sequential steps:

1. Identifying the instructional goal.

   In this step the instructor must determine exactly what the student should be able to do at the completion of the instruction period.

2. Conducting an instructional analysis.

   Here the instructor must identify the subordinate skills that a student must master in order to achieve the above goal. In other words the concept of a goal must be structurally broken down into easier managable parts or modules.

3. Identifying entry behaviors and characteristics.

   In this step the prerequisite skills of the student must be identified. This process of identification should include the consideration of the general maturation background and educational background of the students.

4. Writing performance objectives.

   Based on the instructional analysis and the assessment of entry behaviors and characteristics, the performance expectations, that is, what it is the students will be able to do at the completion of each separate module should be determined.

5. Developing criterion-referenced tests.

   Here, assessment instruments must be developed to measure how well the students have achieved what was described in the objectives.

6. Developing an instructional strategy.

Based on the five preceding steps, a strategy must be developed in which the instructional model will be used to reach the ultimate objective. This strategy should include preinstructional activities, presentation of information, practice and feedback, testing, and follow-through activities.

7. Developing and selecting instruction.

In this step the instructional strategy is used to produce the instructional module which will include all the necessary instructional materials.

8. Designing and constructing the formative evaluation.

A series of evaluations should be conducted on the completed module to determine how effectively the module works and to identify areas where improvement is needed.

9. Revising instruction.

In this final step, data are summarized from various evaluations, the weak areas are re-examined, and the instruction strategy is revised in order to make it a more effective tool (Dick and Carey, 1978).

C. The Design of the Self-Directed Programming Course

The introductory programming course at Simon Fraser University (CMPT 103) is a "semi-self-directed" study course. Although a certain flexibility in work scheduling is possible, it is not a self-paced course. There are specific due dates for assignments, examinations and the term project.

The teaching format of the CMPT 103 course is that of one lecture, and one tutorial session per week. There are also twenty-seven hours of "open lab" per week where tutorial assistants are available to answer student questions on a personal basis.

The thirteen week course includes 10 assignments, a final project, two midterm examinations and one final examination.

The self-directed study aspect of the CMPT 103 course has shown itself to be successful as a teaching strategy for this type of course but the time frame in which the course must be completed, does not allow for self-pacing. As a result, it is necessary to have professional assistance available throughout the course in order to help all students meet assignment deadlines. It is the author's opinion that a similar course with audio-taped lectures and a self-pacing format would alleviate the need for much of the staffing currently allocated to this course.

The survey of the literature showed that self-directed study and methods of study that included the use of audio-taped

instruction can be implemented and used sucessfully. In the area of computer programming, the combination of audio and printed material seemed the best choice. Although at first video-tape lectures seemed attractive, it soon became apparent that this method might require students to try to concentrate on two visual displays simultaneously - the video-tape monitor and the computer terminal. For this reason it seemed best to omit the use of a video component for instructional purposes and to concentrate on the audio material.

The shortage of technically trained personnel to teach computer programming in schools, colleges and universities (Gopaulsingh, 1977) points to the necessity for a self-directed study course. A self-directed study course with lectures prepared and taped by a professional instructor of computer science can lend the expertise necessary to teach computer programming without the teacher in charge of the course being required to have a high level of technical training.

A self-directed study course might also make it possible for individuals who had been "playing around with" home computers to channel their interests into a more directed and disciplined use of the machines.

Familiarity with computers in the home situation could then be augmented by a conscious pursuit of an understanding of what was previously only an intuitive or a trial-and-error procedure.

A self-directed study course allows students to proceed at their own pace and at their own competency level. Students and

instructors alike were found to be more motivated by this type of learning as it allows greater flexibility in their schedules (Simmons, Ward and Thompson, 1974); also the teaching of the course is more natural as the instructor is not in the contrived position of having to 'fill in' a full teaching period. The particular computing language to be taught does not affect the way in which the course is designed. Adaptation of the course to the various other computer languages may be somewhat time consuming but is essentially a straight-forward exercise.

I. Choice of Method

An experiment at the University of Delaware (Hartman and Caroly, 1971) showed the importance of stating behavioral objectives and providing plenty of pertinent examples and exercises to reinforce the learning process. All three concepts were incorporated into the combination of audio tapes and study guide.

The use of tutorial sessions is recommended as this lends a personal aspect to the course (Keller, 1968). In keeping with Keller's philosophy, the audio tapes were designed as vehicles for motivation rather than merely as sources of information.

Audio methods were chosen over audio-visual techniques; in many of the presentations the student would be interacting with a computer terminal and the introduction of a second visual aid would only distract the student from the close observation of the computer terminal.

Modern methods of "top-down structured" programming are used throughout the course. Although PASCAL is used as the programming medium, problem solving and algorithm design are the main emphases of the course.

II. Development of the Course Criteria

As computing courses attract more and more students, the diversity of backgrounds of those students increases. With this diversity and with the range of information in computing science in mind, every possible help should be made available to the students and their instructors.

Although the basic principles of programming can be taught, the only way to learn programming is by doing, and doing implies a considerable amount of self-directed study.

The "Systems Approach Model" (Dick and Carey, 1978) was used as the model for the design of the course instruction. This instruction is implemented on the basis of the following criteria:

1. At the end of the course the student should be able to define a programming problem and produce a suitable computer programmed solution. This program should use the structured programming techniques taught in the course. The student should also have an understanding of structured data types, control statements and procedures.
2. There is no specific prerequisite for this course except that the student should have a general understanding of

simple logic. However this skill can be also acquired within the context of the course.

3. The maturation background and educational background expected of the students is senior secondary school standing or better.

4. A module of the course involves reading a chapter of the study guide, reading the corresponding chapter(s) of the textbook, listening to the corresponding audio tape, and completing the required assignment(s). Performance objectives are directly related to the mastery of the topics presented in the textbook.

   For example, after reading Chapter II of the study guide and Chapter 2 of the textbook, listening to audio tape 2, and completing Assignment 2, the student is expected to understand (at a simplified level) the overall organization of a PASCAL program.

5. The main measure of the student's performance is the manner in which the assignment in each model is completed. However, an instructor may supplement this with quizzes and formal examinations if necessary.

The design of the course is based on recent trends in the theory of programming. The course presents a disciplined approach to programming by concentrating on structured programming and software design. The students are introduced to these programmming methodologies right from the start and thus are not likely to develop bad programming habits.

Structured Programming

The CMPT 103 introductory programming course at Simon Fraser University uses a technique of programming called structured programming. Most of the former techniques of programming are considered to be "unstructured". In "unstructured" programming modular design is almost non-existent and the sequence of program instructions appears to be in random order. Structured programming involves the use of modular design and a fixed sequence of program instructions usually referred to as a "top-down" approach. Because of these techniques, structured programming is considered to be easier to write, easier to understand, easier to fix, and easier to change than an unstructured program. An unstructured program is a program where the instruction sequence is devoid of planned organization. A structured program can be compared to the organization of a book or a thesis. The overall organization is broken into several component topics and each of these is subdivided further as needed.

The basic process of structured programming is the breaking down of a given problem into a series of simpler problems. Each simpler problem is in turn broken down until it becomes a simple mangageable unit, i.e. no further reduction is possible or no further redcuction is necessary. These small units are then implemented as programmed procedures which have simple, easily manageable structures, rather than having to manipulate the

larger complex units from which they are derived. The result of such structuring is usually a clear, concise and workable program.

> "This process represents the most important concept in structured programming, i.e., that a problem can be solved by repeatedly breaking it down into subproblems, until every subproblem can be solved. If you plan this decomposition before you try to write it out in the narrow, precise, and time-consuming syntax of the target language (i.e., the programming language you use to solve the problem), you will have a better chance of getting your program right the first time." (Williams, 1981)

III. Software Design

Software design deals with the design of individual computer programs and their implementation and interaction in a large system. The design of good programs and ultimately good systems has been foremost in the minds of systems designers since the first major software failures of the 1960s.

> This first software crisis accented the fact that the craftmanship approach to programming was not good enough for the development of large-scale systems. It also became apparent that an engineering discipline had to be applied to programming to solve problems such as missed schedules, overruns, unreliability, and low usability. As a result, a search was launched for good definitions of the underlying principles of computing as well as for precise descriptions of the process of software development." (Wasserman et al., 1978).

The development cycle of software can be briefly described as having five phases:

1. Analysis. Here, a review of system specification takes place and the functional performance requirements are laid out. This phase governs the design and development of the

entire project.

2. Design. Here the performance requirements are translated into functional flowcharts and the project is broken down into easily manageable modules. A draft of the computer program development specification is also prepared.
3. Coding and checkout. Here, all program modules are coded and tested; this phase includes tests to make sure that the interaction between each module is according to specifications. All documentation must also be complete by the end of this phase.
4. Test and integration. The software is integrated with the hardware (the computer that the programs will be implemented on) and is formally tested against the requirements set out in the performance specifications. The system is now ready for use by the customer.
5. Operation and support. Even when the above four phases are meticulously carried out, there will still be some errors present in the finished system. During the life cycle of the system these errors must be corrected as they are found and modifications to the original design may need to be carried out (Wasserman et. al., 1978).

Even though this introductory course does not afford the student the opportunity to work with large software systems, the basic rules of software design described above still apply for the most part to the more simplistic programs students are required to produce.

Analysis of the problem, the design of a solution algorithm, coding, and testing are an integral part of all computer programs. In terms of the course described in this thesis, for every program the student is required to:

a. analyze the problem and choose the variables that will be used in the solution algorithm

b. design the algorithm - in this case, draw a flowchart as described in the study guide.

c. write the PASCAL code to represent the flowchart solution. The code must be documented as it is written.

d. type the code into the computer and "run" it to make sure the program works. Test the program with different sets of data to make sure that it will work for many cases.

e. operation and support can only be discussed throughout the course as student programs are not used in an operations environment.

IV. Choice of a Computer Language and a Computer System

PASCAL was chosen as the ideal language for this course because it is a coherent, powerful and well-defined language that has gained wide acceptance (Tiberghien, 1981).

> "The programming language PASCAL was originally developed for teaching programming with emphasis on the techniques known as structured programming. More recently, as programmers in business and industry have begun to discover the limitations of traditional programming languages such as COBOL and FORTRAN, interest in putting PASCAL to work in the world outside

the classroom has increased. An important boost for PASCAL has come from its widespread implementation on microcomputers. PASCAL is now established as one of the major programming languages that every programmer should know" (Graham, 1983).

PASCAL is also a well designed language that encourages the programmer to write clearly so as to make the program "self-documenting" with only a modest need for additional written documentation.

The PASCAL language is one of the few languages which helps the programmer in program design. It aids the programmer in specifying the process and the data clearly and naturally.

The APPLE II microcomputer system was chosen as the computer for the course because its proven ability to support the PASCAL language, the abundance of supplementary materials available for students, the cost of this system compared to other systems on the market and its widespread use throughout North America.

V. Choice of the Textbook

Several textbooks on PASCAL were examined including the textbook used for the CMPT 103 course at Simon Fraser University. This book was not chosen as it was directed primarily toward the scientifically oriented student and assumed some previous programming experience. Other textbooks were rejected for similar reasons, i.e. they were aimed at a very specialized segment of the student population.

Introduction to PASCAL Including UCSD PASCAL by Rodnay Zaks was chosen as it was the specific intention of the author that the book be designed to be read and understood by everyone, whether novice or experienced programmer, who wants to learn how to program in PASCAL.

The arrangements of the chapters take the student from simple concepts to complex ones in a steady progression. The early chapters present the student with those features of the language necessary to understand the concepts of computer programming. The later chapters build upon that which has already been learned thus allowing the student to design and write more complex programs.

VI. Development of the Audio Tapes

The audio tapes were originally developed on a reel-to-reel recorder to ensure a high quality recording and to allow for refined editing facilities. The final versions of each tape were later transferred to cassette tapes which are much easier for students to use on small portable recorders.

The course itself involves three main tasks:

1. mastering the computer system - in this case the APPLE II microcomputer
2. mastering the principles involved in the formulation of solution algorithms
3. the transformation of these algorithms into the actual PASCAL coded instructions which can be understood and

executed by the computer.

The above tasks were broken up into sub-tasks or topics with each tape designed as an introductory lecture on a specific topic.

For example, the first audio tape is a step-by-step series of instructions on how to type a PASCAL program into the computer and how to execute it. Instructions range from how to switch on the machine and insert the diskettes to the simple use of the operating system are covered.

Other tapes, dealing with algorithms or programming examples, refer to specific pages and figures in the textbook and/or the study guide to reinforce the concepts being discussed. Where actual examples of computer programs from the study guide are discussed, the tape addresses the intricacies of the code chosen to implement the solution algorithm. These subtle but important points could easily be missed by students when they are reviewing written examples without the aid of supplementary instruction.

The self-paced aspect of the audio-tape lectures allows the instructor to cover a greater amount of material in much more detail than can normally be covered in a conventional lecture. Students are more likely to question material on the tape than face the potentially embarrassing situation of approaching the instructor with their own incorrectly transcribed lecture notes. Such questioning increases the rate at which the teaching materials can be improved (Lower, 1981).

VII. Development of the Study Guide

The study guide is designed to be the focal point for the audio tapes and the textbook. The theme of well-structured and reliable computer programs is consistently reinforced in each chapter. It is simply not enough to teach the syntax of a programming language to beginning students without formalized instruction on the formation of algorithms and the proper use of syntax. Without such careful, detailed instruction too many bad habits are formed.

The development of the study guide follows the principle that learning is best accomplished when new information is based on previously learned material. The student makes use of recently acquired knowledge to accomplish the task at hand (Winne and Marx, 1977). With this in mind, each chapter of the study guide is structured to include five basic sections:

1. a reading assignment pertaining to specific sections of the study guide and the textbook
2. suggested exercises from the textbook to provide a self-check mechanism to makes sure the student understands the material. Sometimes the exercises will also point out alternate coding methods and important language limitations.
3. a special "Reflect Section" which contains a few questions to help students test their knowledge of the material from several different perspectives which may not have been apparent to them from the readings and the exercises

4. assignments which require the student to work specifically with the new concepts presented in the current readings and taped lectures
5. a section to discuss special techniques and styles related to the current work being covered.

Applying the Dick and Carey Criteria

The Dick and Carey criteria mentioned above were applied to the development of the self-directed study course reported on here. The discussion below notes specifically how each of the Dick and Carey criteria were implemented.

1. Identifying the instructional goal.

   The instructional goal for this course is one that is common to all introductory computer programming courses: "At the end of the course, the student will be able to define a programming problem and produce a suitable programmed solution using structured programming techniques."

2. Conducting an instructional analysis.

   An informal survey was conducted at various learning institutions from elementary school level to university level to determine what prerequisite skills were conditional to begin teaching computer programming concepts. It was found that teachers were able to introduce this discipline at all grade levels where the concept of counting had already been mastered. As this course is intended for the senior secondary school and first year university levels,

further analysys was unneccessary.

3. Identifying entry behaviors and characteristics.

As this course is designed for senior secondary school and first year university students, these students would be expected to be highly motivated, good readers, and good problem solvers.

4. Writing performance objectives.

Performance objectives were stipulated for each module of the course. In terms of the course materials, one chapter of the study guide represents a single module. The study guide consists of eight chapters.

a. CHAPTER 1:

Given a simple programming problem, the student can draw a flowchart solution and describe the five basic components of the computer that are required to implement the solution.

b. CHAPTER 2:

Given a computer program in PASCAL, the student can type to program into the APPLE-PASCAL system and use the "editor" to alter the text of the program.

c. CHAPTER 3:

Given a problem definition, the student can draw a flowchart solution and implement this solution on the APPLE-PASCAL system using basic PASCAL data types.

d. CHAPTER 4:

Given a problem definition requiring knowledge of

advanced PASCAL programming concepts such as looping, the "if-then-else" construct, and the "case" statement, the student can flowchart and program an efficient solution on the computer.

e. CHAPTER 5:

Given a problem requiring that data be sorted into either ascending or descending order, the student can flowchart and implement a solution on the computer.

f. CHAPTER 6:

Given a problem definition requiring the use of modules as a basis for the solution, the student can flowchart and implement a modular solution on the computer using "structured programming" techniques.

g. CHAPTER 7:

Given a problem definition requiring the use of multiple storage locations, the student can flowchart and implement a solution on the computer using arrays as the major data structure.

h. CHAPTER 8:

Given a problem definition requiring the keeping of records, the student can flowchart and implement a solution on the computer using the "Record" data type.

5. Developing criterion-referenced tests.

As the nature of this course requires that the student hand in completed programming assignments on a regular basis, the correctness and style of the student's program is a good and

sufficient test of how the student is progressing. It should be noted that the nature of computer programming is such that the student is not able to hand in an assignment without first gaining a reasonable understanding of the work being done.

6. Developing an instructional strategy.

The instructional strategy is based on research in the area of audio and written instruction as described above in "Survey of the Literature". The course developed in this thesis centres around the study guide through which the use of the textbook and the audio tapes is coordinated. The study guide was chosen as the focal point of instruction as it can be employed to tailor existing materials to specific needs and provide the learning effectiveness and efficiency that may be lacking in existing sources (Langdon, 1978).

7. Developing and selecting instruction.

The instructional strategy of the course dictates the use of a textbook, the study guide and audio tapes. However, depending on the ultimate goal that an instructor may have in mind for the students, this instructional strategy is sufficiently flexible to allow the course to be supplemented by formal or informal lectures in order to expand on the concepts presented in each chapter of the textbook and the study guide.

8. Designing and constructing the formative evaluation.

The course will be taught to different groups with varied

educational backgrounds and goals in order to identify the strengths and weaknesses of the course as they pertain to each group.

9. Revising instruction.

The formative evaluation from test groups will be used to update and improve and if necessary, to revise the instructional strategy. In this way this type of course can serve as a useful design tool now and in the future.

D. Introduction to the Self-Directed Study Course

The programming aspect of this course has been designed to incorporate the new approaches and exercises related to the concept of software design found in the literature. The pedagogy evolved from the self-directed study approaches that were surveyed in the literature, as well as the author's own experience as a teacher of computer programming. The model arrived at through this study is one which makes use of self-directed study complemented by laboratory periods in which the students can get individual help with their problems.

This PASCAL self-directed study course in PASCAL is based on the study guide (see Appendix C) plus the "Introduction to PASCAL Including UCSD PASCAL" textbook by Rodnay Zaks (2d. ed. Sybex 1981).

I. Study Guide

The study guide itself is divided into eight chapters and two appendices and deals with chapters 1 to 11, chapter 15, and the appendices of the textbook. Chapters 12, 13 and 14 of the textbook are not covered in this course as these chapters deal with advanced topics. In the opinion of the author (of this thesis), these topics are best left for a second course.

Assignments and Project

This course is designed to be taught in a thirteen week semester. As it is a self-directed study course, individual instructors may speed up or shorten this time frame to suit individual needs. There are ten assignments in the course. Each assignment covers a specific concept of programming in the PASCAL language. It is suggested that one assignment be completed each week with three weeks given for the completion of a final project.

II. Lectures

There are nine audio-tape lectures covering selected chapters of the study guide and the textbook. The purpose of these tapes is to expand upon the conceptual material presented in the study guide and the textbook and to broaden the scope of the student's understanding of the course material.

III. Teaching Assistants

It is suggested that assignments be marked by teaching assistants who can also be available to answer questions and help solve programming problems in an open laboratory type environment. This arrangement has the advantage of giving the student a variety of approaches to solving problems.

IV. Equipment and Materials

For each student in the course, the following materials are required:

1. one 48K Apple computer with a PASCAL language card installed.
2. two disk drives
3. a TV set or video monitor connected to the Apple computer.
4. The following diskettes:
   a. one "APPLE1:" diskette (to start up the system)
   b. one "APPLE2:" diskette (to use the PASCAL compiler)
   c. one "APPLE3:" diskette (this diskette contains special programs, i.e., FORMATTER - to format a brand new disk.)
5. two blank diskettes (more as needed)
6. one cassette tape recorder
7. five cassette tapes containing a total of nine lectures
8. one textbook, "An Introduction to PASCAL Including UCSD PASCAL" by Rodney Zaks (2d. ed. Sybex 1981)
9. one Apple PASCAL Language Reference Manual
10. one Apple PASCAL Operating System Reference Manual.

The diskette marked "APPLE1:" is needed to start the system. The "APPLE2:" diskette contains the necessary system information to allow the computer to processes PASCAL programs.

E. Assessment of the "New" Course

The course designed as a pilot project for this thesis was tested by two separate groups of students. Each group of students came from a different educational background and had different expectations of a computer course in terms of their own personal needs in the area of computing science and computer programming.

I. Assessment by Computer Science Undergraduate Students

The first group to try the "new" course were undergraduate students in the regular computing science undergraduate program at Simon Fraser University. An assessment was made of the new study guide, taped lectures and the textbook in comparison with the existing study guide, formal lectures and textbook of the CMPT 103 course at Simon Fraser University. Three students from this CMPT 103 course volunteered to do approximately three weeks work using the "new" course and at the end of that course to complete a questionnaire. The work from chapter IV of the study guide which included assignments 4, 5, and 6. was chosen for the assessment as these students were about to undertake similar work in the CMPT 103 course. The students could then compare both types of instruction without losing any time in their own course. The questionnaire was designed to find out whether or not the course presentation was helpful under self-directed study conditions.

Results of Assessment by Computing Science Undergraduate Students

The answers to the questionnaire indicated that the study guide, tapes and textbook were judged to be a considerable improvement over their CMPT 103 counterparts.

There were several inconsistencies in the original versions of the audio tapes and the study guide. On some of the tapes the speed of instruction was too fast. In the study guide some of the material was out of sequence and some figures were incorrectly labelled. All these errors were corrected.

On the positive side the following points were made by the students trying the new course:

a. The instructions on the tape were clear and to the point.

b. The concepts were well presented with plenty of examples to provide for clarity and understanding.

c. The tapes can be used by students to reinforce learning at home, in class, while waiting for a bus, etc. This means that the instructor is always at hand.

d. "Introduction to the APPLE-PASCAL System" was the most important chapter in the study guide. The introduction brings into clear focus the APPLE-PASCAL system and its operations. It saves a lot of the stumbling over the PASCAL system that students in the regular course encounter at the beginning of the semester.

e. The "Reflect Section" in the study guide is an excellent review aid.

A summary of the responses to the questionnaire is given in TABLE ONE on the following page. The following five-point scale was used in soliciting student responses:

Scale:	Not Helpful	Some Help	Adequate	Helpful	Very Helpful
	(1)	(2)	(3)	(4)	(5)

TABLE ONE - UNDERGRADUATE STUDENTS (Questions 1 - 7)

	Not Helpful	Some Help	Adequate	Helpful	Very Helpful
	(Numbers below represent number of students)				
A. Study Guide					
1. How do you rate the CMPT 103 course Study Guide?	2	1	-	-	-
2. How do you rate the "new" course Study Guide?	-	-	1	-	2
B. Textbooks					
3. How do you rate the CMPT 103 textbook?	1	2	-	-	-
4. How do you rate the "new" course textbook	-	-	-	3	-
C. Labs and Tapes					
5. How do you rate the CMPT 103 Labs?	2	1	-	-	-
6. How do you rate the "new" course tapes?	-	1	-	1	1
7. Indicate the helpfulness of each of the following in learning how to use the terminals.					
a) Demonstrations	1	-	1	-	1
b) CMPT 103 Study Guide	2	1	-	-	-
c) "New Study Guide"	-	-	1	-	2
d) Lectures	1	2	-	-	-
e) Tapes	-	1	-	-	2

The following questions require only short answers.

If you would like to add any further comments, please use the last page and say as much as you like.

8. Which part of the "new" course study guide did you find most useful?

STUDENT 1: THE TAPES

THE FLOWCHART EXAMPLES

STUDENT 2: THE TASKS WERE EXPLAINED WELL

THE SALIENT POINTS OF THE ASSIGNMENT WERE DISCUSSED

THE CONTENTS AND EXPECTATIONS FOR THE ASSIGNMENT WERE GIVEN

STUDENT 3: THE EXAMPLE OF THE 'SORT' AND HOW IT WORKED

9. Which part of the "new" course study guide could have been made more useful - and how?

STUDENT 1: CROSS REFERENCE TO OTHER CHAPTERS ON RELATED CONCEPTS COULD BE PRINTED OUT

STUDENT 2: EVERY FOURTH ASSIGNMENT SHOULD BE LARGER THAN THE OTHERS IN ORDER TO INCORPORATE ALL THE SMALLER CONCEPTS LEARNED IN THE PREVIOUS PARTS

STUDENT 3: THE STATEMENT OF THE ASSIGNMENTS COULD HAVE BEEN MORE EXACT

10. What comments do you have about:-

a. the CMPT 103 text book?

STUDENT 1: TOO MATH ORIENTED
NOT EASY TO READ
DISCUSSION OF FILES WERE POOR

STUDENT 2: INAPPROPRIATE FOR TEACHING INTRODUCTORY PASCAL

STUDENT 3: THE TEXTBOOK WAS GOOD EXCEPT FOR 'PROCEDURES' AND 'FUNCTIONS'

b. the "new" course text book?

STUDENT 1: EASIER READING - APPEARS TO BE STRAIGHT FORWARD
FLOWCHARTS AND PASCAL SYNTAX CHARTS ARE INCLUDED AS VISUAL AIDS

STUDENT 2: GOOD IMPROVEMENT OVER THE EXISTING TEXT
AT TIMES I THOUGHT IT MOVED A BIT QUICKLY

STUDENT 3: VERY GOOD, TAKES YOU THROUGH EXAMPLES
EASIER TO READ

11. What comments do you have about the CMPT 103 labs?

STUDENT 1: LAB T.A.s WERE NOT FAMILIAR WITH PASCAL

STUDENT 2: ACCESS TO T.A.s WAS DIFFICULT

STUDENT 3: NOT ENOUGH REFERENCE MATERIAL FOR THE APPLES

12. What comments do you have about the "new" course tapes?

STUDENT 1: PROVIDES REVIEW OF CONCEPTS, REINFORCEMENTS AND AUDITORY LEARNING

STUDENT 2: I HAVE FOUND THE TAPES EXTREMELY USEFUL. THEY REALLY WOULD BE INVALUABLE IN THE LABS

STUDENT 3: I THOUGHT IN GENERAL THE TAPES WERE GOOD BUT A LITTLE TOO SHORT - THE ONE WHERE A 'SORT' WAS WORKED THROUGH WAS VERY GOOD

13. Would you please compare the relative usefulness (helpfulness) of the CMPT 103 labs and the "new" course tapes.

STUDENT 1: AT PRESENT THE T.A. LACK OF FAMILIARITY WITH PASCAL SYNTAX AND SYSTEMS OPERATIONS PROVIDE VERY LITTLE BENEFIT

THE TAPE OBVIOUSLY PROVIDES A SOUND WAY OF IMPARTING KNOWLEDGE AND DIRECT ACCESS TO INFORMATION

WITH T.A.s YOU HAVE TO LINE UP AND YOUR WAIT MAY BE FRUITLESS; WITH TAPES YOU REPLAY UNTIL YOUR UNDERSTANDING IS CLEAR

STUDENT 2: THE NEW COURSE TAPES ARE MORE USEFUL THAN THE PRESENT LAB SETUP

STUDENT 3: THE LABS WERE PRETTY MUCH USELESS - I FOUND THE COMBINATION OF THE PRINTED MATERIAL AND THE TAPES VERY GOOD

14. Is there a question I should have asked you about these two courses?

STUDENT 1: HOW HELPFUL WAS THE INSTRUCTOR CURRENTLY TEACHING THE COURSE COMPARED TO THE TAPE?

ANSWER: THE TAPED INSTRUCTIONS COME FROM THE VOICE OF EXPERIENCE AND EXPERTISE IN THE SUBJECT-MATTER

STUDENT 2: SHOULD THE MARKING EMPHASIS REMAIN THE SAME?

STUDENT 3: I THINK THAT IN THE COURSE I TOOK TOO MANY NEW CONCEPTS WERE INTRODUCED TOO QUICKLY

15. If there were two things in the "new" course you would not want to change, what would they be?

STUDENT 1: 1. THE TAPES

2. THE FORMAT OF THE STUDY GUIDE

STUDENT 2: TAPES

STUDY/TEACHING GUIDE

STUDENT 3: THE TEXTBOOK, THE TAPES, THE APPLE MICROCOMPUTERS

16. If there were two things in the "new" course you would want to change, what would they be?

STUDENT 1: TEACH 'PROCEDURES' EARLIER IN THE COURSE

LESS ASSIGNMENTS

STUDENT 2: LECTURE SHOULD BE TREATED MORE LIKE A TUTORIAL

STUDENT 3: MORE INFORMATION ON HOW TO USE THE APPLES

HAVE A GOOD INTRO TO THE APPLES

If you have any further comments, or want to enlarge on any of the previous ones, please use the back of this page.

STUDENT 3: AS SELF STUDY, MORE REFERENCE MATERIAL IS NEEDED

II. Assessment by Graduate Students in the Faculty of Education

The second group of students to try the "new" course were graduate students in the Faculty of Education at Simon Fraser University. The "new" course was introduced to these students as part of a basic introductory course to general computer science with emphasis on programming the APPLE II microcomputer using PASCAL as the programming language. Although the "new" course is designed as a self-directed study course, it was decided to supplement the first official offering of it with lectures in order to offset the typographical errors and organizational errors that are usually discovered in the first offering of many courses.

It should be pointed out that this was the first time this type of course had ever been offered to graduate education students at Simon Fraser University and the expectations of these students turned out to be quite different from the expectations of the computing science undergraduate students. Although CMPT 601 (the graduate student course) was supposed to consist mainly (approximately 70%) of computer programming with some discussion on the effects of computers in society, the students initially perceived the course as offering equal amounts on both topics. The course was intended to be offered

to novices only but the actual enrolment consisted of seven novices, four students with some previous programming experience, and three students who had already completed the CMPT 103 course at Simon Fraser University. As a result, this instructor decided to break the course up into three separate groups, Group 1, Group 2, and Group 3 respectively, according to the above separations. For the programming requirements of the course the first two groups worked on the material of the course described in this thesis. The novices (Group 1) worked only on the course material described in this thesis. The more advanced group of four students (Group 2) started their work at chapter III of the study guide and completed the course with some advanced work. The "final project" was not required in order to leave room for the "computers in society" aspect of the course. The students who had completed CMPT 103 (Group 3) were taught advanced programming techniques and did not participate in the materials described in this thesis.

Breaking the class up into three different groups had the effect of making each group a separate course each with its own instruction periods. However, since the allocated instruction time was in reality only enough for a single course, each group had to make do with only a third of the instruction that had been originally planned for them. The result of this strategy was that the social implications aspect of the course was largely ignored in favour of computer programming which now dominated up to ninty-five per cent of the instruction time,

much to the dissatisfaction of the students.

Results of Assessment by Graduate Students in Education

The questionnaire was given to the Groups 1 and 2 described above. Ten out of a total of eleven students in the both groups responded. (Group 3 did not use the course materials developed in this thesis.) The answers to this questionnaire indicated that the study guide, tapes and textbook were adequate but not sufficiently comprehensive to provide the self-confidence that a student wishes to achieve in a programming course. Throughout the questionnaire the message was that more examples are needed in the study guide, the tapes and the textbook.

Most students felt the earlier chapters of the study guide and the early sessions on the tapes were good. These early chapters and first tape sessions brought the students along at a very slow pace compared to the usual CMPT 103 course for Computing Science undergraduate students. However, it seemed that as the course progressed and the pace picked up, these graduate Education students felt the amount of instruction was insufficient.

A summary of the questionnaire as completed by the Graduate Education Students is given on the following pages.

1. Are you planning to use what you have learned in CMPT 601 in the school system? YES (8) NO (2)

If yes (check all appropriate boxes);

a. for own use only (7)

b. for administrative work (4)

c. as a teacher of computing (3)

Questions 2, 3 and 4 are shown in TABLE TWO on the following page. The five-point scale from the undergraduate questionnaire (shown below) was used again here.

Scale:	Not Helpful	Some Help	Adequate	Helpful	Very Helpful
	(1)	(2)	(3)	(4)	(5)

TABLE TWO - GRADUATE STUDENTS (Questions 2 - 4)

	Not Helpful	Some Help	Adequate	Helpful	Very Helpful
	(Numbers below represent number of students)				
2. How useful was the Study Guide? (Disregard irrelevant things such as typos, etc.)	-	4	5	1	-
3. How useful was the textbook?	-	6	-	4	-
4. How useful were the Audio Tapes?	2	5	3	-	-

5. Which chapters of the study guide were most useful?

STUDENT 1: CHAPTER 1, CHAPTER 4, AND APPENDIX B.

STUDENT 2: CHAPTERS 1 TO 3.

STUDENT 3: CHAPTER 1 - FLOWCHARTING SYMBOLS, CHAPTER 4, AND CHAPTER 5.

STUDENT 4: THE BEGINNING CHAPTERS.

STUDENT 5: (NO COMMENTS)

STUDENT 6: THE FLOWCHART EXAMPLES.

STUDENT 7: THE BEGINNING CHAPTERS.

STUDENT 8: (NO COMMENTS)

STUDENT 9: THE INTRODUCTION AND CHAPTER 1.

STUDENT 10: (NO COMMENTS)

6. Which chapters were least useful?

STUDENT 1: CHAPTER 2.

STUDENT 2: CHAPTER 4.

STUDENT 3: ALL THE CHAPTERS WERE USEFUL - SOME MORE USEFUL THAN OTHERS AS MENTIONED IN QUESTION #5 ABOVE.

STUDENT 4: THE LAST ONES.

STUDENT 5: THE RECORD PROGRAM ON PAGE 89, THE INTRO COMMANDS TO GET INTO THE SYSTEM, E.G., PAGE 20 BOTTOM AND PAGE 21 BOTTOM NEED EXAMPLE AND SPECIFIC INSTRUCTIONS.

STUDENT 6: (NO COMMENTS)

STUDENT 7: LATER CHAPTERS HAD TOO FEW EXAMPLES.

STUDENT 8: (NO COMMENTS)

STUDENT 9: THE LATER CHAPTERS - ESPECIALLY CHAPTER 8.

STUDENT 10: (NO COMMENTS)

7. Comments on the study guide

STUDENT 1: GENERALLY HELPFUL BUT MANY OF THE POINTS COVERED WERE NOT DESCRIBED IN SUFFICIENT DETAIL;
GREATER REFERENCE TO THE APPLE II IN PARTICULAR IS NEEDED.

STUDENT 2: WEAK ON EXPLANATION OF STORAGE LAYOUTS AND THEIR RELATIONSHIP TO THE FLOWCHARTS;
INPUT/OUTPUT EXAMPLES CONFUSING;
SOME MATERIAL OUT OF SEQUENCE.

STUDENT 3: THE STUDY GUIDE IS A GOOD OVERVIEW OF THE PASCAL LANGUAGE. IF USED WITH THE TEXTBOOK AS IT IS INTENDED, I BELIEVE IT CAN SERVE AS A USEFUL TOOL. A FEW MORE EXAMPLES WOULD BE HELPFUL.

STUDENT 4: THE STUDY GUIDE COULD BE USEFUL IF PERTINENT DETAIL IS SUPPORTED BY EXAMPLE. IT COULD HANDHOLD THE STUDENT THROUGHOUT THE COURSE.

STUDENT 5: FOR A COMPUTER COURSE FOR EDUCATORS, SOME PROBLEMS SHOULD HAVE BEEN BASED ON EDUCATIONAL APPLICATIONS.

STUDENT 6: GOOD FOR CMPT 103 BUT NOT A COURSE FOR EDUCATORS.

STUDENT 7: MORE EXAMPLES WOULD BE MOST USEFUL. SHORT EXAMPLES THAT WE COULD REFER TO WOULD BE USEFUL.

STUDENT 8: SOME PROGRAMMING ERRORS.

STUDENT 9: THE EARLIER CHAPTERS ARE GOOD BECAUSE THEY EXPLAIN THE NEW MATERIAL THAT IS PRESENTED AND PROVIDE THE STUDENT WITH SOME BACKGROUND AND EXAMPLES TO USE AS A REFERENCE. THE LATER CHAPTERS DO NOT AND CONSEQUENTLY ALL STUDENTS IN THE COURSE HAVE SPENT A TREMENDOUS AMOUNT OF TIME SCRAMBLING FROM ONE STEP TO THE NEXT. NEEDS AN EXAMPLE OF CREATING AND READING A FILE.

STUDENT 10: THE REQUIREMENTS FOR INPUT, I.E., ONE LINE OR MORE, AND OUTPUT, I.E., ONE LINE OR MORE, SHOULD BE MORE EXPLICIT.

8. Which chapters of the textbook were most useful?

STUDENT 1: CHAPTER 6 - MOST FREQUENTLY REFERRED TO.

STUDENT 2: CHAPTERS 2, 3, 4 AND 6.

STUDENT 3: ALL CHAPTERS EXCEPT CHAPTER 11.

STUDENT 4: ABOUT THE SAME

STUDENT 5: (NO COMMENTS)

STUDENT 6: ALL BUT THE ONE ON PARAMETERS ETC.

STUDENT 7: CHAPTERS 1-5.

STUDENT 8: (NO COMMENTS)

STUDENT 9: ALL GENERALLY OK.

STUDENT 10: (NO COMMENTS)

9. Which chapters of the textbook were least useful?

STUDENT 1: CHAPTER 12 - LEAST FREQUENTLY REFERRED TO.

STUDENT 2: CHAPTER 7.

STUDENT 3: CHAPTER 11.

STUDENT 4: ALL ABOUT THE SAME.

STUDENT 5: (NO COMMENTS)

STUDENT 6: THE CHAPTER ON PARAMETERS.

STUDENT 7: LATER CHAPTERS HAD VERY LONG COMPLICATED PROGRAMS.

STUDENT 8: (NO COMMENTS)

STUDENT 9: THE CHAPTER ON FILES.

STUDENT 10: (NO COMMENTS)

10. Comments on textbook

STUDENT 1: EACH CHAPTER WAS OF VALUE AS THE MATERIAL WAS BEING STUDIED.

STUDENT 2: NO GOOD EXPLANATION OF SYNTAX DIAGRAMS.

STUDENT 3: VERY WELL WRITTEN.
FILES ARE NOT WELL EXPLAINED.

STUDENT 4: LACKS SPECIFIC DETAILS.
NEED PROGRAM SO THAT ONE CAN SEE HOW TO PUT OPERATION IN PROGRAM.

STUDENT 5: (NO COMMENTS)

STUDENT 6: USEFUL, BUT NOT ENOUGH EXAMPLES.

STUDENT 7: MORE EXAMPLES WOULD BE HELPFUL.

STUDENT 8: (NO COMMENTS)

STUDENT 9: FOUND PROGRAM EXAMPLES GENERALLY CONFUSING.
WOULD PREFER A MORE DIRECT REFERENCE TO UCSD PASCAL.

STUDENT 10: CUT DOWN ON NUMBER OF QUESTIONS AND SHOW ANSWERS TO ALL PROBLEMS. TOO MANY TYPOGRAPHICAL ERRORS.

11. Comments on supervision in the Lab.

STUDENT 1: (NO COMMENTS)

STUDENT 2: THE TUTORIAL ASSISTANT WAS HELPFUL.
SOME INSTRUCTION MIGHT BE USEFUL TO WORK THROUGH BASIC PROCEDURE FOR EACH ASSIGNMENT AS WE HAVE RECENTLY DONE IN LECTURES.

STUDENT 3: THE TUTORIAL ASSISTANT WAS HELPFUL.

STUDENT 4: GOOD.

STUDENT 5: OK.

STUDENT 6: GOOD.

STUDENT 7: I RARELY SEEMED TO BE IN THE LAB. WHEN THERE WAS SUPERVISION.

STUDENT 8: GOOD.

STUDENT 9: THE TUTORIAL ASSISTANT DID AN EXCELLENT JOB.

STUDENT 10:

WOULD BE HELPFUL TO HAVE THE INSTRUCTOR IN THE LAB. AS WELL AS THE TUTORIAL ASSISTANT.

12. Comments on assignments

STUDENT 1: WOULD HAVE PREFERRED TO HAVE MORE POINTED DISCRETE ASSIGNMENTS, PERHAPS MORE SINGLE-CONCEPTED IN ORDER THAT LANGUAGE DETAILS MIGHT BE PRACTICED. THEN, PERHAPS, MORE EXTENSIVE ASSIGNMENTS MIGHT BE UNDERTAKEN.

STUDENT 2: SCOPE OF ASSIGNMENT WAS FINE, BUT SOME CONFUSION RESULTED FROM TRYING TO USE TECHNIQUES THAT WERE TOO ADVANCED TOO EARLY. STUDY GUIDE SHOULD GIVE MORE GUIDANCE ON THIS.

STUDENT 3: NEED A FEW MORE ONE-CONCEPT ASSIGNMENTS BEFORE ASSIGNMENTS 8, 9 AND 10.

STUDENT 4: SOME HAD CONCEPTS BEYOND THE KNOWLEDGE OF THE PROGRAMMER.

STUDENT 5: WOULD PREFER EDUCATIONAL APPLICATIONS FOR AT LEAST SOME PROBLEMS.

STUDENT 6: PERHAPS MORE EXAMPLES TAYLORED TO PEOPLE WHO WILL HAVE TO TEACH PROGRAMMING.

STUDNET 7: WE NEEDED INSTRUCTION ON THE ASSIGNMENTS BEFORE SPENDING HOURS DECIDING WHAT SHOULD BE DONE.

STUDENT 8: DIFFICULT.

STUDENT 9: SHOULD BE EDUCATION ORIENTED EXAMPLES!!!
-DISCUSSION AND EXAMPLES OF HOW TO PROTECT AND PROCESS USER INPUT TO PREVENT PASCAL FROM "BOMBING".
-SOME ASSIGNMENT INSTRUCTIONS NEED TO BE CLEARER AND HAVE HINTS FOR PROGRAMMING DESIGN.

STUDENT 10: (NO COMMENTS)

13. Comments re. supplementary materials

STUDENT 1: (NO COMMENTS)

STUDENT 2: TAPES GENERALLY NOT HELPFUL. PERHAPS THEY SHOULD BE FOCUSSED MORE SPECIFICALLY ON THE ASSIGNMENTS.

STUDENT 3: AUDIO TAPES ARE A BIT TOO GENERAL.

STUDENT 4: (NO COMMENTS)

STUDENT 5: TEXTBOOK BY LUEHRMANN & PECKHAM WAS OFTEN USEFUL.

STUDENT 6: (NO COMMENTS)

STUDENT 7: IT WAS DIFFICULT TO KNOW WHICH TO DO FIRST: TRY THE ASSIGNMENT, READ THE TEXT, READ THE STUDY GUIDE, OR LISTEN TO THE TAPES.

STUDENT 8: (NO COMMENTS)

STUDENT 9: THE AUDIO TAPES DID NOT ADDRESS THE PROBLEMS THAT OCCURED IN THE PROGRAMS.

STUDENT 10: (NO COMMENTS)

14. Comments on the course

a. generally

STUDENT 1: I FOUND THE COURSE TO BE LABOUR INTENSIVE AND WAS FREQUENTLY FRUSTRATED BY THE AMOUNT OF TIME REQUIRED TO FULLY UNDERSTAND A PROGRAMMING PROBLEM. PRIOR DISCUSSION OF PROGRAMMING TECHNIQUES WOULD HAVE BEEN HELPFUL.

STUDENT 2: I SPENT A GOOD DEAL OF TIME IN A RATHER FRUSTRATED STATE DUE TO MISINTERPRETATIONS BOTH ON MY PART AS WELL AS BECAUSE OF THE INPUT/OUTPUT EXAMPLES IN THE STUDY GUIDE.

STUDENT 3: IT COULD HAVE BEEN AN EXCELLENT COURSE IF IT WAS GEARED TOWARDS A SINGLE GROUP RATHER THAN THREE GROUPS AT ONCE.

STUDENT 4: TRIED TO COVER A LARGE AREA. SHOULD FOCUS ON ONE ASPECT.

STUDENT 5: SHOULD NEVER HAVE BEEN SPLIT INTO GROUPS. SET UP COURSE, THEN ATTENDEES EITHER LIKE IT, LUMP IT, OR LEAVE IT.

STUDENT 6: A GREAT DEAL OF WORK OF WHICH I FELT I WAS FLOUNDERING.

STUDENT 7: THE COURSE WAS TOO ORIENTED TO PROGRAMMING.

STUDENT 8: I DON'T FEEL THAT MY NEEDS AS AN EDUCATOR WERE MET.

STUDENT 9: FOUND THE CLASS TIME OF MARGINAL VALUE. BEST CLASSES WERE THE STRUCTURED ONES. ON PROCEDURES, ARRAYS AND RECURSION. MOST OTHER CLASSES HAD NO APPARENT PLANNING AND CAME ACROSS AS VERY DISORGANIZED - A DANGEROUS THING TO DO WHEN TEACHING TEACHERS!

STUDENT 10: THE MOST ABSORBING COURSE I HAVE EVER TAKEN. I'M HOOKED ON COMPUTERS.

b. compared to other <u>computer programming courses</u> you may have taken

STUDENT 1: PERHAPS A LITTLE HEAVIER THAN MOST.

STUDENT 2: <u>MUCH</u> MORE COMPREHENSIVE THAN U.B.C. EDUC 217 (BASIC). WE TALKED ABOUT MANY THINGS BUT GOT INTO LITTLE DETAIL.

STUDENT 3: THE INSTRUCTOR EXPLAINS HIS SUBJECT MATTER VERY WELL. I ENJOYED THE COURSE.

STUDENT 4: SAME AS OTHERS - CONFUSING.

STUDENT 5: UNIMPRESSED - BUT LIKELY DUE TO THE SPLITTING UP OF GROUPS. ALSO I FEEL THAT THIS COURSE FOR WORKING TEACHERS SHOULD BE SPREAD OVER TWO SEMESTERS WITH CLASSES MEETING ON ALTERNATE WEEKS. THIS WOULD GIVE STUDENTS TIME TO DIGEST THE MATERIAL AND TO COMPLETE ASSIGNMENTS WITHOUT MAKING COMPUTING SCIENCE THEIR TOTAL LIFE.

STUUDENT 6: I LEARNED MORE ABOUT PROGRAMMING IN CMPT 601 (THIS COURSE) THAN IN CMPT 103 (THE BEGINNING COUSE FOR COMPUTER SCIENCE STUDENTS). IN CMPT 103 I LEARNED HOW TO PASS THE COURSE BUT NOT MUCH ABOUT PROGRAMMINC.

STUDENT 7: (NO COMMENTS)

STUDENT 8: LESS THAN SATISFACTORY.

STUDENT 9: TAKEN NONE.

STUDENT 10: (NO COMMENTS)

15. If there were two things in the course you would not want to change, what would they be?

STUDENT 1: THE ASSIGNMENT WERE HELPFUL AND SHOULD REMAIN.

STUDENT 2: THE SCOPE OF PROGRAMMING EXPERIENCE.

STUDENT 3: THE ASSIGNMENTS.

LACK OF A FINAL EXAMINATION.

STUDENT 4: PASCAL AS A LANGUAGE.

LAB ASSISTANCE.

STUDENT 5: THE NUMBER OF ASSIGNMENTS.

LAB TIME.

STUDENT 6: THE INSTRUCTOR.

COMPUTER TIME.

STUDENT 7: THE COURSE OUTLINE.

PROGRAMMING FINAL PROJECT.

STUDENT 8: FEEDBACK ON ASSIGNMENTS WAS GOOD.

THE STUDY GUIDE WAS SOMEWHAT HANDY.

STUDENT 9: SORRY BUT EVERYTHING NEEDS CHANGING. COURSE PROPOSAL WAS GOOD BUT THE IMPLEMENTATION SURELY DESTROYED IT. MUST HAVE A TUTORIAL ASSISTANT.

STUDENT 10: THE PROBLEMS.

THE DOCUMENTATION.

16. If there were two things in the course you would want to change, what would they be?

STUDENT 1: GREATER EMPHASIS GIVEN TO EDUCATIONAL USES AND APPLICATIONS.

MORE PERTINENT LECTURE TOPICS.

NO MORE ATTEMPS AT STREAMING FOR ABILITY AND BACKGROUND.

STUDENT 2: LIMIT THE COURSE TO ONE TYPE OF STUDENT.

STUDENT 3: NO GROUPING.

STUDENT 4: PRESENTATION OF CONCEPT MORE DIDACTIC.

MORE SAMPLE PROGRAMS (SIMPLE).

STUDENT 5: LECTURES SHOULD BE SCHEDULED ON DAYS WHEN HOLIDAY WILL NOT CAUSE THEM TO BE CANCELLED.

STUDENT 6: LAB OPEN HOLIDAY AND SUNDAYS.

NO SPLITTING OF CLASS INTO GROUPS.

STUDENT 7: DEAL WITH SHORTER PROGRAMS (ASSIGNMENTS) THAT COULD BE WORKED WITH AGAIN LATER ON IN THE COURSE.

KEEP A GROUP WORKING TOGETHER SO THERE IS INTERACTION AMONG THEM BY GIVING INSTRUCTION BEFORE AN ASSIGNMENT IS STARTED, AFTER IT IS STARTED, AND JUST BEFORE IT IS FINISHED.

STUDENT 8: LECTURES WOULD BE MORE INSTRUCTIVE.

LABS WOULD BE MORE DIRECTED.

STUDENT 9: STRUCTURED CLASS TIME WOULD HELP A LOT WITH SPECIFIC DISCUSSION OF NEW TOPICS BEFORE THE ASSIGNMENT.

STUDENT 10: HAVE ONE GROUP ONLY.

DISCUSSION OF EACH ASSIGNMENT BEFORE STARTING IT.

17. Are there any portions which could have been omitted altogether?

STUDENT 1: NO.

STUDENT 2: "COMPUTERS IN SOCIETY" SHOULD BE A SEPARATE COURSE.

STUDENT 3: "COMPUTERS IN SOCIETY" TEXTBOOK WAS OUTDATED.

STUDENT 4: COMPUTERS IN SOCIETY.

STUDENT 5: (NO COMMENTS)

STUDENT 6: AS IT WAS.

STUDENT 7: THE EXAMINATIONS.

STUDENT 8: (NO COMMENTS)

STUDENT 9: NO.

STUDENT 10: (NO COMMENTS)

18. Is there a question I should have asked you about which you would like to comment.

STUDENT 1: LIGHTENING THE WORKLOAD WOULD BE BENEFICIAL.

STUDENT 2: (NO COMMENTS)

STUDENT 3: CONSIDERING THE VARIETY OF STUDENTS IN THE CLASS YOU DID WELL.

STUDENT 4: COURSE SHOULD BE DIRECTED TO ONE GROUP.
DISCUSS MORE CONCEPTS.
USE MORE EXAMPLES.

STUDENT 5: (NO COMMENTS)

STUDENT 6: NO.

STUDENT 7: (NO COMMENTS)

STUDENT 8: (NO COMMENTS)

STUDENT 9: SHOULD DISCUSS HOW TO SET UP THE BASIC SHELL OF AN ASSIGNMENT.
MORE DISSCUSSION ON HOW TO ORGANIZE AN ASSIGNMENT.

STUDENT 10: I THINK THE COURSE WAS TOO LONG GIVEN THE TWO HOLIDAYS WE MISSED CLASSES AND THE FACT THAT THERE WAS THREE GROUPS.
MORE INSTRUCTION NEEDED ON TAPES AND IN CLASS

F. Summary

I. Analysis and Comparison of the Two Student Questionnaires

In summarizing the results of the two sets of questionnaires, certain background information needs to be taken into consideration. In the first group, the undergraduate students who were taking the course and using the material from the thesis were individuals whose main interest was in computing science as a discipline, perhaps as a tool, but chiefly as a discipline. These students seemed to be quite happy with the way the course was set up and with the information presented. They found the combination of the tapes and the study guide to be very useful. Although they did not seem to have as much faith in some of the work done in the labs, they were helped by the tapes to work on their own and do much more of their own problem solving. None of them had had previous courses in computing which is of interest when a comparison is made with the second group of students who were taking the course.

The second time that the course was presented, it was given to graduate students in the Faculty of Education. These students varied in their reasons for taking the course: some wanted it for their own use only, some were using it for administrative purposes, and the rest were intending to teach computing courses at the junior or secondary level. Some of

these graduate students had had previous courses in computing, but only one of the students who answered the questionnaire (STUDENT 6) had taken a computing course at Simon Fraser University. As a group they found the course difficult; they wished to have more instruction; they wished to have more information about the examples and wanted more help in working out the actual problems of the course.

It would seem in comparing these two groups of students, that there may be a need for a special type of course for those whose specific intention is to teach computing techniques in the regular school system. The orientation of this thesis was toward a pilot project to see what could be done by combining the use of a textbook, a study guide and a series of tapes. For the computer science students this seems to be a very useful method, but for the graduate students in education, this course may need some minor changes.

There may have been some problem too with the way in which the idea of the course was presented to these education students. It seems that they felt they were enrolling in a course which was to be a general overview of computing in education, (with very little time assigned to a "hands-on" type of activity) and a considerable amount of discussion of how computing as a science relates to everyday living. This really does not add much to the consideration of whether or not the original idea of the thesis was borne out. The original idea of the thesis was to find out whether or not a course could be

taught as a mainly self-directed type of activity. The conclusion that can be drawn is that for those people who are interested in becoming computer programmers and are perhaps intending to take further courses, the tapes, the study guide and the textbook make a reasonable combination and will allow these students to satisfy most of their course needs.

For those individuals who are working within the school system and who are likely to have expectations of a "methods" approach to the subject matter, this course may not be as effective. It should be pointed out that the teachers who were taking the course tended to be critical of the actual pedagogy of the CMPT 601 course in general and the way in which the work was structured, rather than with whether or not they were learning how to do the programming exercises which were assigned.

The make-up of a course for university students is likely to be somewhat different from a course taught at the junior or secondary level. For these levels there is not the same amount of work to be covered, there is less depth of explanation and there is more opportunity and time in which to present the course material. There is also considerably more time available for discussion of particular items at the secondary school level. Perhaps the people in the second group who are professional teachers may have missed the point of what was being done. They may have thought that they were into a "methods" course rather than taking a course for their own

introduction to another discipline.

This summary would indicate that there is room for the development of a different type of course for education professionals. It would likely be a course which they themselves would be able to present to their students at the junior and secondary levels in the regular school system. From the point of view of courses in which the student is learning the basic principles of programming in order to be able to do programming, it would seem that the current combination of textbook, tapes and study guide are satisfactory. If in the future this course were given to a similar group of professional educators, this difference in orientation would have to be very carefully spelled out to them in advance.

The official description of the CMPT 103 course is:

> "INTRODUCTION TO A PROGRAMMING LANGUAGE FOR COMPUTING SCIENCE MAJORS/MINORS/HONORS - This course introduces the Computing Major/Minor/Honors student to a programming language. Programming assignments cover techniques such as looping, decision making, construction of subroutines, input/output handling and documentation. Emphasis will be given to teaching the student techniques of structured programming."

The way the CMPT 103 course is taught at present differs chiefly in terms of the use of tapes. The course makes use of lectures and labs. together with the textbook and study guide. The course proposed in this thesis would make use of a text and a study guide, but would supplement these with tapes. The tapes would make it possible for a student taking the course to work much more independently and rely less on lectures and lab. instruction.

The CMPT 103 course was designed with the expectation that most of the students who graduate from the course will major in computing science. The approach therefore is to have the students start problem solving at a very early stage. They are to solve as much of the problem as they can on their own in preparation for the more advanced theory they will be studying in the future. For those who are not intending to work toward a degree in computer science, the idea is still to guide them toward eventually becoming systems analysts - not just programmers or coders.

This thesis has shown, however, that there is a need for both a service course and a technical course in which the student can learn the very basic details of programming without being taught to design a very large computing system - a series of interrelated programs. Such a course would require considerably more instruction in the classroom and many more worked-out examples in the textbook, the study guide and the audio tapes. Only a small amount of original work would be required of the latter student.

II. Conclusions

This thesis has addressed directly and indirectly several questions which should be considered when developing a self-directed computer programming course, particularly:

a. Is the development of a good computer program dependent mainly upon the understanding of basic concepts - the theory?

b. Can the understanding of concepts be replaced adequately by training - 'how to' - which does not emphasize concepts but relies on practice and experience?

c. What would be a good mix of theory and application in teaching a beginning programmer?

The basic premise for the development of the course is that the theory of problem-solving techniques such as analysis of the problem, stepwise specification of a solution algorithm and finally the structured-language implementation of the solution algorithm can be taught. However, is this knowledge retained by the student without the benefit of the immediate applications of the principles learned?

d. If such application of the principles is necessary, how much practice is needed and should that practice be massed or distributed?

The survey results give some answers to reflect upon. It should however, be recognized here, that each question could be a thesis topic in its own right. In relation to this thesis, the

following observations were made:

a. Those students who were interested in becoming computer programmers made good use of the material which set out the basic concepts - the theory of programming. The graduate education students on the other hand were not interested in becoming programmers. From discussion with these students during the course it seems that they were interested basically in knowing how to teach programming and were somewhat impatient with other aspects of the course.

b. There isn't a clear cut answer to the question of training vs education because the students involved are working at a university level and were likely to ask why, not just how. In other words it seems that university students are past the point of being simply trained; they need to know the reasons behind the training.

c. With this as a background it is possible to go on to question (c) and consider what would be a good mix of theory and application in teaching a beginning programmer.

The questionnaire results clearly indicate that two different approaches were involved. The three students who wanted to learn programming were pleased with the emphasis on theory and the opportunity to put the theory into practice but on a limited scale of practice.

The students who were in the education courses seemed to be much more interested in a methods orientation. Most expressed a desire to know how to teach the subject of computer science in general rather than being concerned with the discipline itself. These students wanted to know what the computer does - not the theory of computer programming. Being professionally trained educators rather than computer programmers, they wanted more information about how to teach the subject. This was a reasonable request as they were not interested in giving up the discipline of pedagogy for the discipline of computer science.

d. As for mass vs distributed practice, having taught many computing programming courses including the "new" course described in this thesis, the author feels that practice in the application of the principles is more sucessfully taught in a distributed manner with a gradual building from the known to the unknown rather than with large amounts of theory being presented at one time.

The use of tapes makes it possible to implement this idea of distributed practice as the student can go back and review the material and attempt additional problems which will help in the understanding of the basic principles involved.

G. Recommendations for Further Study

As the thesis involves developmental work, this new PASCAL course will need to be closely examined to determine its worth compared with traditional methods of teaching. An experimental course could be designed in which groups can be matched to compare various combinations of teaching methods to find out which methods achieve the best and most lasting results. Experiments could be conducted to see how different groups work using the ordinary classroom instruction, using the textbook plus tapes and study guide, or using all of them as a unit.

In the three years the author has been teaching CMPT 103, the course has grown from 250 students to 800 students. The growth in class size reflects the current infatuation with computers. This type of growth has made it possible to assess the effectiveness of various approaches but only in a subjective way. The pressure of the sudden increase in enrolment has not allowed time for real experimentation and assessment in depth.

Some large academic departments have made it compulsory for their students to take at least one computing course. The result is very large classes composed of two quite distinct groups of students - those who want to learn programming and those who must take a computer course. The problem of selecting appropriate textbooks, lecture material, exercises and examinations becomes difficult under these conditions.

There is however more flexibility in the human element involved; instructors and teaching assistants can adjust more readily to the new demands.

Some of the administrative problems involve finding enough qualified people to serve as instructors and lab. assistants, providing sufficient terminals and other mechanical requirements, and suitable study and work space for students.

Another problem which is not too evident at the present time but will become increasingly apparent, is boredom when students who have had access to computer instruction either at home or in the regular school system are asked to do work which they have already covered. Along side these somewhat experienced students will be those who, while intelligent, are almost completely naive about computers and computer science.

Fortunately, most of the students who are in computing courses are also in courses which are relatively structured - commerce, mathematics, etc. The arts student who come into the course may have difficulty, but are usually aware of the difference of approach between the science courses and the arts courses.

This discussion of the problems of the current beginning programming course at Simon Fraser University points out the need for more than one approach to the teaching of such an introductory computing science course.

A recommendation would be the development of three different courses at the introductory level. One course would be for those students for whom computer science is a discipline; in this course the emphasis would be on the theory and principles of computer programming with sufficient exercise material to allow the principles to be applied. The expectation is that for these students there would be further courses in computer science and this preliminary course would be the foundation material for them.

A second course would be designed quite frankly for students who are taking a course because it is compulsory. For such students the emphasis would be on how computers fit the kind of work they are likely to be doing. There would be emphasis on general principles but not to the same depth as in the previous course. There would also be some practice and the orientation would be toward making it possible for these students to work with programmers and to understand what may or may not be expected of computers in everyday use.

The third course would be designed for students who are in a Faculty of Education and who wish to know enough about computing science to be able to teach it at a more basic level, such as one would find in junior or secondary schools.

The course designed for this thesis is related to the first group of students mentioned above, but the tapes would be an ideal way of supplementing the regular course material to make it more compatible with the needs of each of the three groups.

REFERENCES

1. Allen, Michael. "Computer Managed Instruction." Journal of Research and Development in Education, v14, n1 (1980), 33-40.

2. APPLE PASCAL Language Reference Manual. Cupertino, California: Apple Computer Inc., 1980.

3. APPLE PASCAL Operating System Reference Manual. Cupertino, California: Apple Computer Inc., 1980.

4. Barringer, Robert, and Haluk Bekiroglu. "The Effects of Audio Tapes On Graduate Student Performance and Attitudes." AEDS Journal, v11, n2 (1978), 38-48.

5. Bork, Alfred. Learning With Computers. Bedford, Mass.: Digital Equipment Corporation, 1981.

6. Davies, W.J.K. Alternatives to Class Teaching in Schools and Colleges. London, Englang: Council for Educational Technology, 1980.

7. Dick, Walter, and Lou Carey. The Systematic Design of Instruction. Glenview, Illinois: Scott, Foresman and Co., 1978.

8. Dijkstra, E.W. "Programmming Considered as a Human Activity." In Proc. IFIP Congress 1965, North-Holland Publishing Co., Amsterdam, The Netherlands, 1965, 213-21.

9. Dijkstra E.W. "A Constructive Approach to the Problem of Program Correctness." BIT, 8, 3 (1968), 174-186.

10. Dijkstra E.W. "Goto Statement Considered Harmful." Comm. of the ACM, II, 3 (March 1968), 147-148, 538, 541.

11. Dijkstra E.W. "Structured Programming." Software Engineering Techniques, Ed. J.N. Buxton & B. Randell. Brussels, Belgium: NATO Scientific Affairs Division, 1970, 84-88.

12. Donaldson, J. R. "Structured Programming." Datamation, v19, n12 (1973), 52-54.

13. Drumheller, Sydney J. Handbook of Curriculum Design for Individualized Instruction: A Systems Approach. Englewood Cliffs, New Jersey: Educational Technology Publications, 1971.

14. DuBoulay, B. & O'Shea, T. How to Work the LOGO machine. University of Edinburgh: Department of Artificial Intelligence, Paper No. 4, 1976.

15. DuBoulay, B., O'Shea, T. & Monk, J. The Black Box Inside the Black Box: Presenting Concepts to Novices. University of Edinburgh: Department of Artificial Intelligence, Paper No. 133, 1980.

16. Esbensen, Thorwald. Working With Individualized Instruction. Belmont, California: Fearon Publishers, 1968.

17. Fisher, Kathleen M. and Brian MacWhinney. Audio-Visual Communication Review, 24 (1976), 229-261.

18. Gibbons, Maurice. Individualized Instruction. New York: Teachers College Press, 1971.

19. Goldschmid, Barbara, and Goldschmid Marcel. Individualizing Instruction In Higher Education: A Review. McGill University, Montreal, Canada: 1972.

20. Gopaulsingh, Karam. A Survey of the Use of Computers in the Secondary Schools of British Columbia and the Development of a Curriculum Guide for an Introductory Course in Computing, unpublished M.A.(Ed) Thesis, Simon Fraser University, 1977.

21. Graham, Neill. Introduction to PASCAL, 2d ed., New York: West Publishing Company, 1983.

22. Hartman, Lyle G.; Behr, Caroly. "A Videotaped Course Designed to Teach Fortran." Audiovisual Instruction, 16, 2 (1971), 33-34.

23. Jensen, Randal R. "Structured Programming." Computer, 14, 3 (1981), 31-50.

24. Keller, Fred S. "Good-bye Teacher..." Journal of Applied Behavioral Analysis, v1, n1 (1968), 79-89.

25. Langdon, Danny G. "The Adjunct study guide." The Instructional Design Library. Englewood Cliffs, New Jersey: Educational Technology Publications, 1978.

26. Lower, Stephen K. "An Audiotutorial Approach to the Teaching of Physical Chemistry and Electrochemistry." Journal of Chemical Education, v58, n10 (1981), 773-776.

27. Mayer, Richard E. "Psychology of Computer Prgramming for Novices." Series in Learning and Cognition. Report No. 81-2. Santa Barbara: University of California, 1981.

28. Mayer, Richard E. "Analysis of a Simple Computer Programming Language: Transactions, Prestatements, and Chunks." Series in Learning and Cognition. Report No. 79-2. Santa Barbara: University of California, 1979.

29. Papert, Seymour. Final Report on the Brookline LOGO Project, Part II: Project Summary and Data Analysis. Artificial Intelligence Memo No. 545. Cambridge, Artificial Intelligence Lab: Massachusetts Institute of Technology, 1979.

30. Russell, James D. "The Audio-Tutorial System." The Instructional Design Library. Ed. Danny G. Langdon. Englewood Cliffs, New Jersey: Educational Technology Publications, 1978.

31. Schneider, Michael G., Steven W. Weingart, and David M. Perlman. An Introduction to Programming and Problem Solving With PASCAL. New York: John Wiley & Sons, 1978.

32. Simmons, Dick B. Darrell L. Ward, and Jack L. Thompson. "Introductory Computer Programming by Extension." Journal of Educational Data Processing, 11, 6 (1974), 19-26.

33. Tiberghien Jacques. The PASCAL Handbook, Berkeley, California: Sybex, 1981.

34. Wasserman, A. I. et. al. "Software Engineering: The Turning Point." Computer, v11, n9 (1978), 30-41.

35. Williams, Greg "Structured Programming and Structured Flowcharts" Byte, v6, n3 (1981), 20-34.

36. Winne, Philip H.; Marx, Ronald W. "Reconceptualizing Research on Teaching." Journal of Educational Psychology, v69, n6 (1977), 668-678.

37. Young Kenneth C. "Using a Computer to Help Implement the Keller Method of Instruction." Educational Technology, 14, 10 (1974), 53-55.

38. Zaks, Rodnay. Introduction to PASCAL Including UCSD PASCAL, 2d ed., Berkeley, California: Sybex, 1981.

BIBLIOGRAPHY

1. Atkinson, Laurence. PASCAL Programming. New York: John Wiley & Sons, 1980.

2. Brainerd, Walter S., Charles H. Goldberg, and Jonathan L. Gross. PASCAL Programming: a Sprial Approach. San Francisco: Boyd & Fraser Publishing Co., 1982.

3. Burnett, James "Self-Paced Fortran." Educational Research and Methods, v11, n2 (1979), 53-56.

4. Conway, Richard, David Gries, and David B. Wortman. Introduction to Structured Programming Using PL/I and SP/k. Cambridge, Mass.: Winthrop Publishers Inc., 1977.

5. Cooper, Doug, and Michael Clancy. Oh! PASCAL!. New York: W. W. Norton & Co., 1982.

6. Dahl, O.J., Dijkstra, E.W., and Hoare, C.A.R. Structured Programming. London, England: Academic Press, 1972.

7. Dunn, Walter L. "Integration of Three New Teaching Techniques in an Introductory Computer Course." ASEE, United States Naval Academy, Annapolis, Maryland, 1977.

8. Eisner, Elliot W. The Educational Imagination, On the Design and Evaluation of School Programs. New York: Macmillan Publishing Co., Inc., 1979.

9. Enos, Judith C.; Van Tilburg, R.L. "Software Design." Computer, 14, 2 (1981), 61-83.

10. Fingar, Peter "The Use of Transparencies for Teaching Computer Concepts." AEDS Monitor, 15, 4/5/6, Oct./Nov./Dec. (1976), 4-7.

11. Heines, Jesse M. "Evaluating Interactive, Computer-Managed Instruction." Annual Conference of the Association for the Development of Computer Based Instructional Systems. San Diago, California, 1979.

12. Heines, Jesse M. "The use of Computer-Managed Instruction to Control On-Site Self-Instructional Training in a Small Systems Customer Environment." Annual Meeting of the Association for the Development of Computer Based Instructional Systems. Dallas, Texas, 1978.

13. Hume, J.N.P., and R. C. Holt. USCD PASCAL: A Beginners Guide to Programming Microcomputers. Reston, Virginia: Reston Publishing Co., 1982.

14. Jensen Kathleen and Niklaus Wirth. User Manual and Report, 2d ed., New York: Springer-Verlag, 1974.

15. Johnson, Mildred Fitzgerald "Computer Literacy: What is it?" Business Education Forum, v35, n3 (1980), 18-22.

16. Kaufman, Roger Emanuel. A Fortran Coloring Book. Cambridge, Massachusetts: The M. I. T. Press, 1978.

17. Kraft, Phillip. Programmers and Managers: The Routinization of Programming in the United States. New York: Springer-Verlag, 1977. 2d ed., New York: Springer-Verlag, 1974.

18. Lewis, William E. Problem-Solving Principles for PASCAL Programmers: Applied Logic, Psychology, and Grit. New Jersey: Hayden Book Co., Inc., 1981.

19. Lynch, Robert E., and John R. Rice. Computers: Their Impact and Use - Structured Programming in PL/1. New York: Holt, Rinehart, and Winston, 1978.

20. Montanelli, Richard G., Jr. "Evaluating PLATO in the Teaching of Computer Science." Journal of Computer-Based Instruction, v5, n3 (1979), 72-76.

21. Nelson, Harold "Logo For Personal Computers." Byte, v6, n6 (1981), 36-44.

22. Nievergelt, Jurg. "Interactive Systems for Education: The New Look of CAI." IFIP Second World Conference on Computer Education. Marseilles, France, 1975.

23. Pogue, Richard E. "The Authoring System: Interface between Author and Computer." Journal of Research and Development in Education, v14, n1 (1980), 57-68.

24. Pohm, A.V.; Smay, T.A. "A Tutorial Overview: Top Down System Design." Computer, 14, 6 (1981), 65-68.

25. Raskin, Jef; Whitney, Tom. "Perspectives on Personal Computing." Computer, 14, 1 (1981), 62-73.

26. Richards, James L. Pascal. New York: Academic Press, 1982.

27. Roid, G. H. "Selecting CAI Author Languages to Solve Instructional Problems." Educational Technology, 14, 5 (1974), 29-31.

28. Seidel, Robert J. "An Heuristic Meta-Model for Computer-Managed Instruction." Journal of Research and Development in Education, v14, n1 (1980), 16-32.

29. Scherer, Donald R. "Backward Chaining: An Effective Means of Teaching Computer Programming." AEDS Monitor, 15, 1/2/3, Jul./Aug./Sep. (1976), 4-5.

30. Schneider, G. M., and S. C. Bruell. Advanced Programming and Problem Solving with PASCAL. New York: John Wiley & Sons, 1981.

31. Shortt, Joseph, and Thomas C. Wilson. Problem Solving and the Computer: A Structured Concept with PL/1(PL/C). Menlo Park, California: Addison-Wesley Publishing Company, 1976.

32. University of Iowa: Weeg Computing Center, Proceedings of 1978 Conference on Computers in the Undergraduate Curricula. 9th, Denver, CO, 1978.

33. Waksman, Abraham "The Introduction of Programming to Prospective Authors of CAI Programs." Educational Technology, 14, 5 (1974), 33-34.

34. Wong, Edmond K. The Development and Refinement of Guidelines for the Use of computer assisted instruction in Secondary Schools, unpublished M.A.(Ed) Thesis, Simon Fraser University, 1974. Burnaby, British Columbia, 1974.

APPENDIX A - AUDIOTAPE LECTURES

Five cassette tapes containing nine audiotape lectures are attached to this thesis.

(nine lectures on 5 cassette tapes are attached to this document)

PASCAL Programming With UCSD PASCAL

A Self-Directed Study Course

Shane Caplin

Simon Fraser University

Preface

As computing courses attract more and more students, the diversity of backgrounds of those students increases. With this diversity and with the range of information in computing science, every possible help should be made available to the students and their instructors.

Although the basic principles of programming can be taught, the only way to learn programming is by doing. And doing implies a considerable amount of self-directed study.

This course has been designed to incorporate new approaches and exercises to the concepts and problems presented in most textbooks. It emphasises the self-directed study approach and is best complemented by laboratory periods in which the students can get individual help with their problems.

Table of Contents

Introduction

This PASCAL self-directed study course is based on this study guide plus the "Introduction to PASCAL Including UCSD PASCAL" textbook by Rodnay Zaks.

Assignments, Project and Examinations

This is a self-paced course. It is designed to give you maximum flexibility in determining your work schedule. Your instructor will determine the length of time in which you are to complete the course and when material is due.

Examinations are based on the material covered in the lectures and the exercises in the study guide and the textbook.

You are responsible for:

1) All material covered during lectures

2) Submission of all assignments and the final project

3) Sitting for both a final written examination and a final oral examination

Midterm examinations will be given at certain times (to be announced) throughout the semester as preparation for the final examinations. The purpose of the midterms is to allow the student to assess his or her understanding of the concepts and material covered at key intervals throughout the course. The course instructor will determine the method of evaluation.

Lectures

There are nine audio tape lectures covering selected chapters of the study guide and the textbook. The purpose of these tapes is to expand upon the conceptual material presented in the study guide and the textbook. and will serve to broaden the scope of the students understanding of the course material.

Open Laboratory

There will be several open laboratory periods each week. Laboratory attendance is not mandatory but you should attend two or three sessions a week. Self-directed study does not mean doing everything by yourself. Do as much as you can on your own and then ask for help in the labs.

Teaching Assistants

Your assignments will be marked by the teaching assistant to whom you have been assigned, but during the lab periods you may get help from whichever teaching assistants are on duty. This arrangement has the advantage of giving you a variety of approaches to solving your problems.

Study Guide

The study guide is divided into eight chapters and two appendicies and deals with chapters 1 to 11, chapter 15, and the appendices of the textbook.
Chapters 12 to 14 will not be covered in this course.
As the sequence of problems is not always the same in the study guide as in the textbook, it is best to check the study guide carefully.

Each chapter of the study guide is divided into parts A,B,C and D, with sometimes an additional Part E.

Note that the chapters in the study guide are designated by roman numerals -I,II,III, etc., and the chapters in the textbook by Arabic numerals - 1,2,3, etc.

Part A

Part A is a reading assignment, in which you will be told to carry on reading the study guide or to read certain portions of the textbook.

Part B

Part B suggests exercises from the textbook which should provide a self-check mechanism to make sure you have learned the material. Sometimes the exercises will also point out alternate 'coding' methods and important 'statement' limitations.

Some of the exercises are of the short quiz type, others may require an extensive amount of work. You are not expected to attempt all the exercises. Do as many as are necessary to gain a firm grasp of the chapter material.

The exercises may also be used as a source of help if you make a note of any questions or alternative solutions that you would like to have checked in the lab. periods.
Midterms and final examinations are based on the exercise problems.

Appendix L of the textbook contains answers to selected ODD-NUMBERED exercises.

Part C

Part C consists of a 'reflect' section which contains a few questions to help you test your knowledge of the chapter, or to bring specific points to your attention.

Part D

Part D contains the assignment. The way in which the assignment is to be prepared and handed in is set out in detail in appendix B of the study guide.

Your T.A.'s comments on each assignment are used as a teaching technique, so it is best to hand in one assignment at a time to avoid losing marks by repeating errors.
Check with your instructor if you do not plan to follow this pattern.

Remember the staff are there to help you, so make use of their assistance.

Part E

Part E includes material which is specific to a given chapter but which doesn't fit into any of the other parts.
This part may also contain additional assignments.

The next section is an introduction to the APPLE-PASCAL system. If you are already familiar with the APPLE-PASCAL system turn to chapter I of the study guide.

Introduction of the APPLE-PASCAL SYSTEM

The purpose of this writeup is to explain briefly how PASCAL works and to provide you with a short overview of the APPLE-PASCAL system followed by a detailed description of an actual terminal session. This writeup is intended to get you started with your first session on the terminal and contains only a very small portion of the many APPLE-PASCAL system commands available.

In order to make use of the full power of APPLE-PASCAL, you will need to read the APPLE-PASCAL Operating System Reference Manual as well as the APPLE-PASCAL Language Reference Manual.

The APPLE-PASCAL system can be thought of as an electronic system which stores information in the form of files, the same way you would store information in a file cabinet. That information may be an essay, a business document, or in this case, a PASCAL program.

How does APPLE-PASCAL work?

The APPLE is a micro-computer - a small computer designed to fit into a very compact space.

Like any other computer, large or small, its function is to process very simple instructions. Although the instructions are simple, the way all computers accomplish their work is through a complex sequence of these simple instructions.

Depending on the computer language chosen, instructions can be processed at various levels, from simple to complex.

In order to process instructions, every computer has several thousand storage locations which are used to:

a. store the instructions

b. store the data relating to those instructions.

For example, the computer had stored the instruction "ADD 2 + 2", it would also have to store both of the 2's as data.

The basic hardware of the computer responds only to "low-level" machine language instructions. As machine language is extremely tiresome for human beings to work with, most people write instructions for computers in "high-level" languages such as PASCAL. The instructions are then translated into a low-level language (machine language) which the basic hardware can understand.

The instructions for doing the translation from the high level language to machine language are themselves usually written in machine language. These special machine language instructions are known collectively as the compiler.

Even though the work is slow and tedious, there is a good reason for communicating with the compiler in machine language. The resulting code (a term for any high-level or low-level computer instructions) is far more efficient than a code derived by first writing the instructions in a high level language.

As the compiler will be used time and time again to translate your high level language, it is important that it be as efficient as possible.

Some Technical Explanations

1. APPLE is a computer

2. PASCAL is a high-level language

3. The computer responds to a low-level (machine) language only

4. You will work with PASCAL, a high-level language

5. Sets of instructions in any language (high-level or low-level) are called programs

6. There is a "go-between" between your PASCAL instructions and the necessary machine language. This "go-between" is a program called the compiler. The PASCAL compiler translates your high-level language PASCAL to a low-level machine language

7. The machine language resulting from the compilation of a high level language is usually called object code. In the case of PASCAL it is called P-Code

8. A person who writes instructions for a computer is called a programmer

The APPLE-PASCAL Files

A file is any collection of statements , commands, data, etc. which you type on the terminal. In order to make use of this way of 'filing', it is important to understand the APPLE-PASCAL disk filing system.
APPLE-PASCAL files are divided essentially into three groups: system files, permanent files and work files.

System Files

These are files containing all system programs and are used to:

a. prompt you at the terminal and interpret your response, i.e. they control the E(DIT mode, F(ILER mode, etc.

b. process your PASCAL program, i.e. C(OMPILE and R(UN modes.

Do not worry about "E(DIT, F(ILER, C(OMPILE and R(UN, etc. at this point. They will be discussed in detail later.

Permanent Files

These are files which you create and store under names of your own choosing. You may work on them by "G(ETing" a copy into your workfile, making the necessary changes and re-saving the file.

Workfiles

During any given session you may create or work on more than one file. Your workfile is a copy of the permanent file on which you are currently working. The G(ET command in F(ILER mode will put a copy of any permanent file into the workfile.

When you choose a permanent file to be worked on as your workfile, the original of that file is left unchanged. If you want to incorporate the workfile changes into the original file, you must re-save it.

This is analagous to the process of making a photocopy of a file in the file cabinet, making some corrections (this step may involve retyping the file) and replacing the original file with the updated version. If the updated version for some reason is not put in the file cabinet, then the original file still remains as the current version. In other words:

- If you start with a permanent file called INFO you use a copy of it as a workfile, leaving the original in the "file cabinet".

- If you do not make changes to the file called INFO in your workfile, there is no need to go through the S(AVE routine.

- If you do make changes in the workfile and want to keep these changes you have two options:

    1) Give the change file a new name e.g. NEWINFO and S(AVE it. This now gives you two permanent files, INFO and NEWINFO.

    2) If you don't want to keep the original INFO file, S(AVE the changed version, keeping the name INFO. This destroys the old version of INFO and puts the changed one in its place.

File names may not be longer that 15 characters including the ".TEXT" suffix.

Read the "Introduction and Overview" sections of the APPLE-PASCAL Operating System Reference Manual before going on to the following Sample Terminal Session.

A Sample Terminal Session

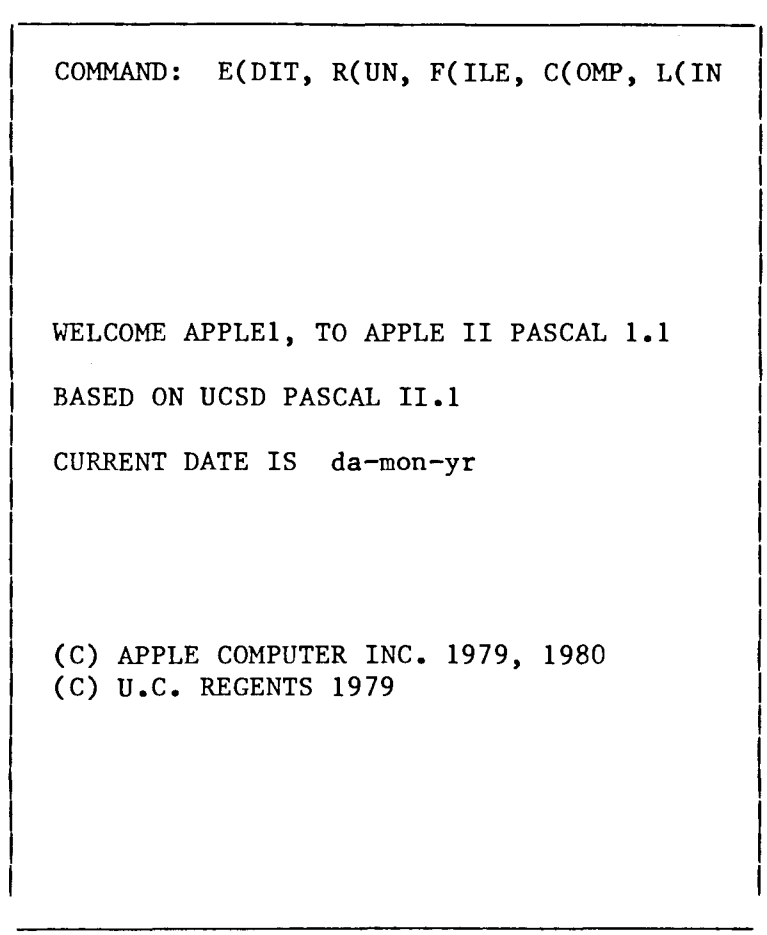
1. Place the diskette marked Apple1 APPLE-PASCAL into the disk drive marked with a 1 (This is also known as VOLUME #4).

2. Place the diskette marked Apple2 APPLE-PASCAL into the bottom disk drive marked with a 2 (This is also known as VOLUME #5).
   (It is possible to use APPLE-PASCAL with only a single disk drive although this is not recommended. Your instructor will advise you how to proceed with these two steps if only one disk drive is available.)

3. Turn on the Video Monitor (As there are several different types of monitors, see your instructor if this step presents any difficulty.)

4. With your left hand, reach around the back of the computer (this is the body on which the monitor usually sits) and flip the switch to "ON".
   The "POWER" light on the keyboard will light up and the computer will come to life with a loud "beep". At this point you will also see the red monitor light of the top disk (VOLUME #4) light up and the disk will start to rotate.

This process of turning on the computer is called "booting the system". In approximately seventeen seconds the following display will appear on the video screen:



The "APPLE1" in "WELCOME APPLE1" may be changed to your own name and "da-mon-yr" may be set to the current day, month and year.

At this point you should begin to read the APPLE-PASCAL Operating System Reference Manual and become familiar with the various modes and levels of the APPLE-PASCAL commands, i.e., E(DIT, R(UN, F(ILE, etc.

Consider the problem of being invited to exhibit a painting at the local art gallery only to find that your allotted space was half the width of your painting. One solution might be to cut the picture in half from top to bottom and display each half separately. The APPLE-PASCAL system has a similar problem to which they have applied the same solution.

APPLE-PASCAL allows 80 characters of PASCAL code to be typed on a single line but the video monitor can display only 40 characters per line. To solve this problem APPLE

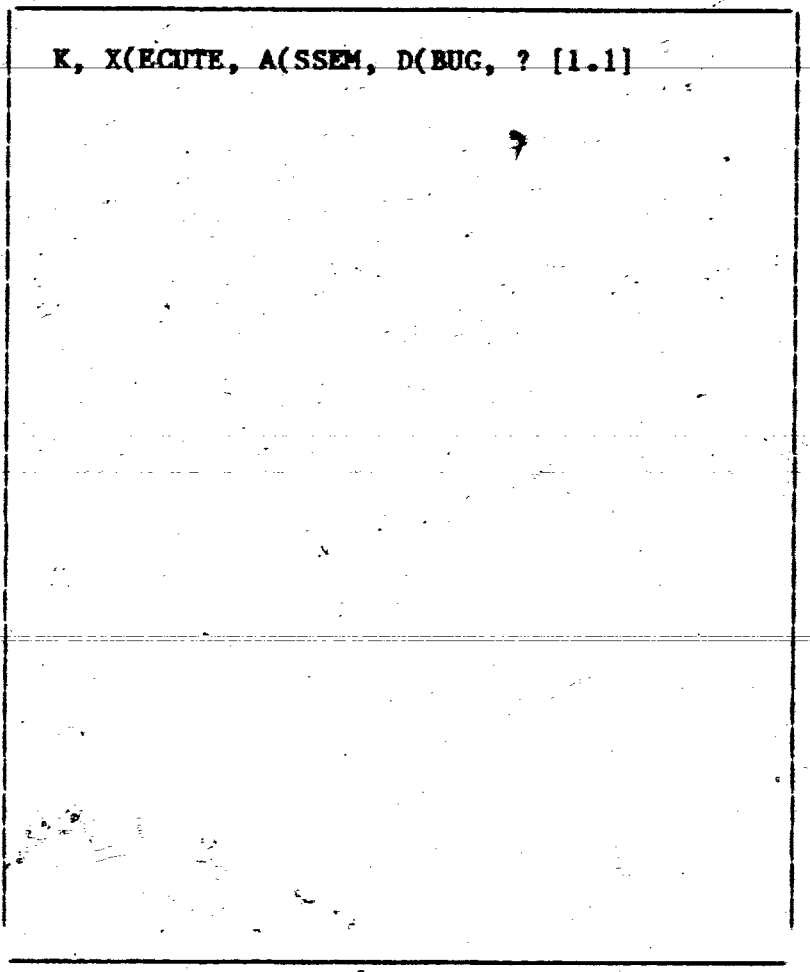
has introduced the concept of the "half screen". When we "boot" the APPLE-PASCAL system we see only the first 40 characters of each line (the left hand side of the screen).

To see the second half (right hand side), while holding down the key marked "CTRL" press the key marked "A".

YOU MUST HOLD DOWN THE "CTRL" KEY FIRST BEFORE PRESSING THE "A" KEY!

This feature will be referred to hereafter as "CTRL-A". During the course of your work with the APPLE-PASCAL system, you will become familiar with other control sequences such as "CTRL-X", "CTRL-C", etc. Remember to always hold down the "CTRL" key first before pressing the second key.

Now you should see this:



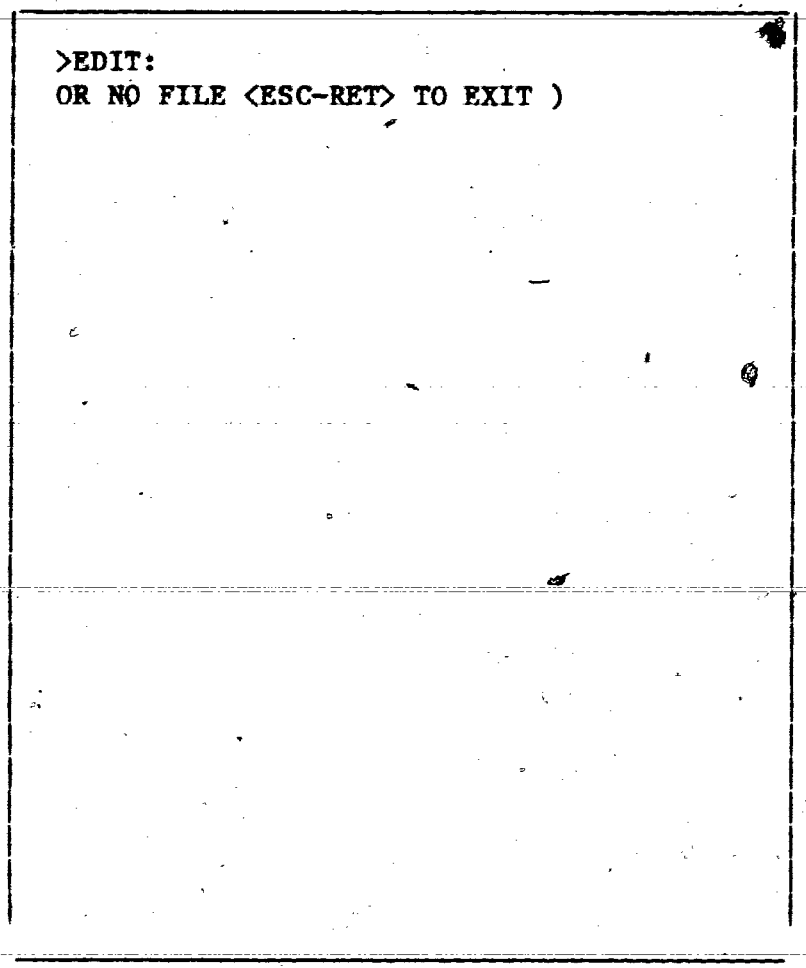
Pressing "CTRL-A" again will take you back to the left side of the screen.

5. The first step in writing a PASCAL program is to create a file to put it in. Although there are several ways to do this, the simplest one is to go directly into E(DIT mode by pressing "E" on the keyboard. If a workfile (refer to the APPLE-PASCAL Operating System Reference Manual if you are still not familiar with the concept of a workfile) already existed, it would now be "loaded" (made available for editing).

If you are using a brand new APPLE1 disk and do not yet have a workfile, when you enter the E(DIT mode, the screen will display the following information:

```
>EDIT:
NO WORKFILE IS PRESENT. FILE? ( <RET> FO
```

The right hand side of the screen will show the rest of the message if you press "CTRL-A".



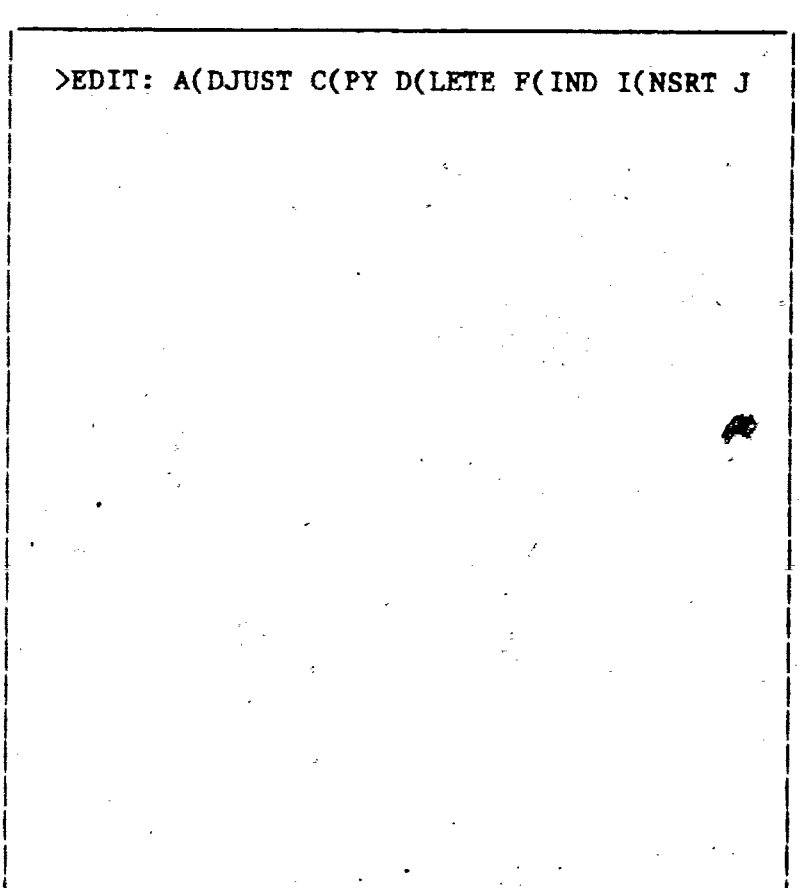
Having seen the rest of the message it would be a good idea to get back to the left hand side of the screen as this is usually the side most worked upon (Press "CTRL-A" again).

The message on the screen tells us that we have two choices:

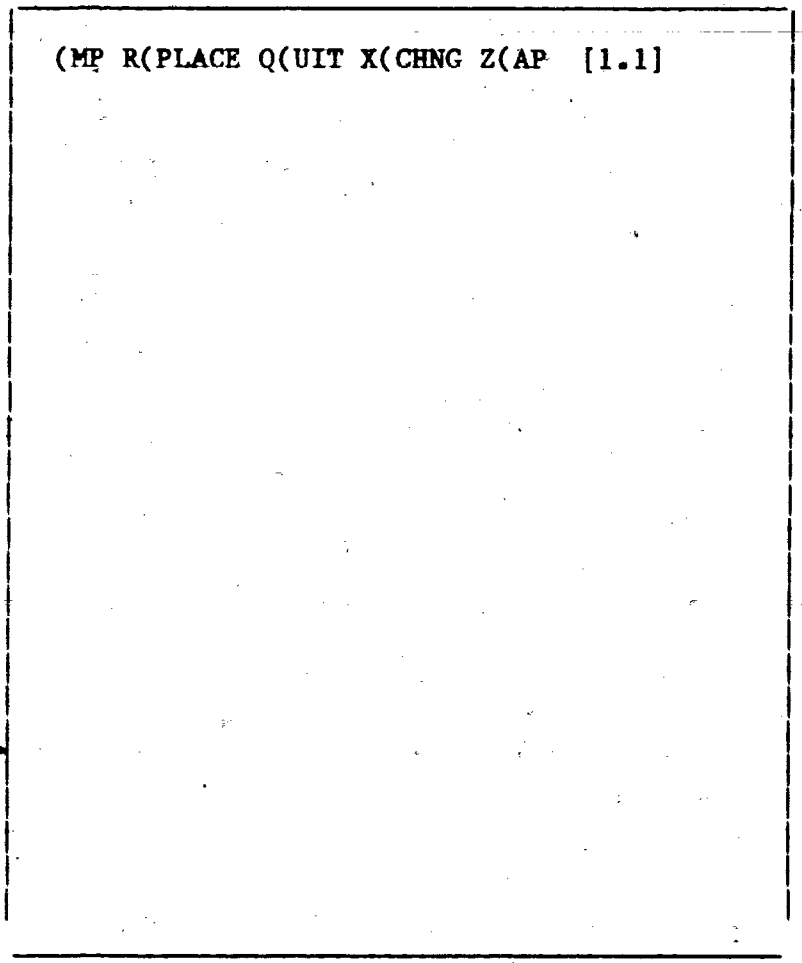
a. Press the "RETURN" key to stay in E(DIT mode or

b. Press the "ESC" (escape) key followed by the "RETURN" key to leave E(DIT mode and return to the "outermost" command level we were at before.

Since we want to write a PASCAL program, press the "RETURN" key (only) to stay in E(DIT mode. Having done this you will notice that there are a new set of commands belonging to the E(DIT mode:

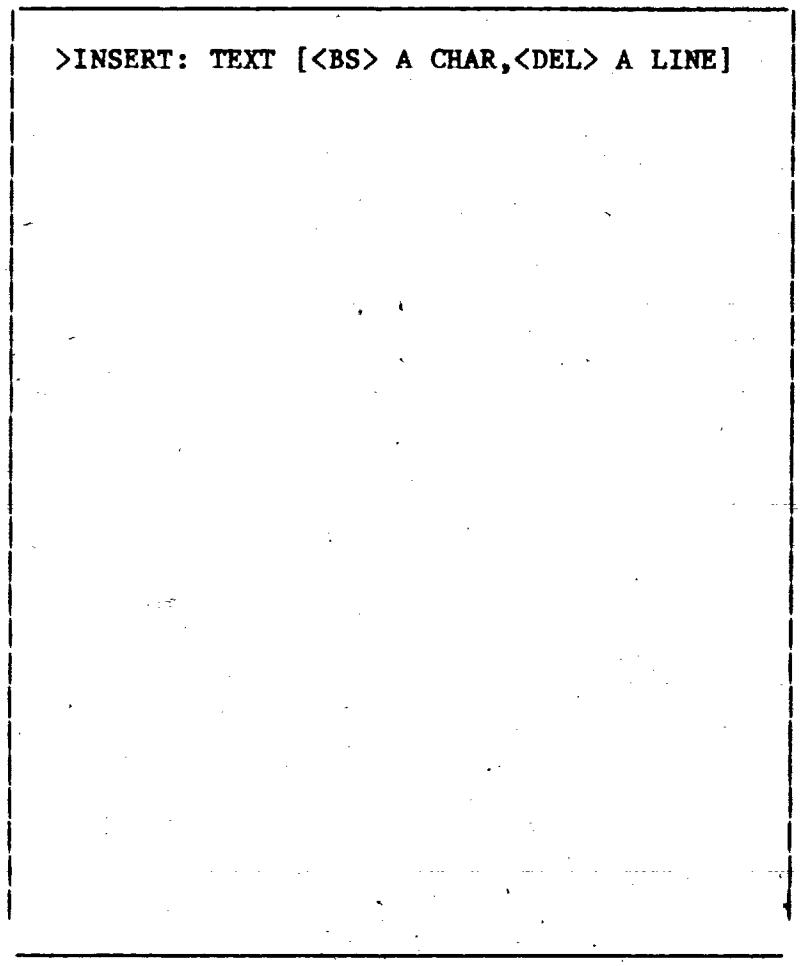


The right hand side of the screen will show the rest of the commands (Press "CTRL-A"):

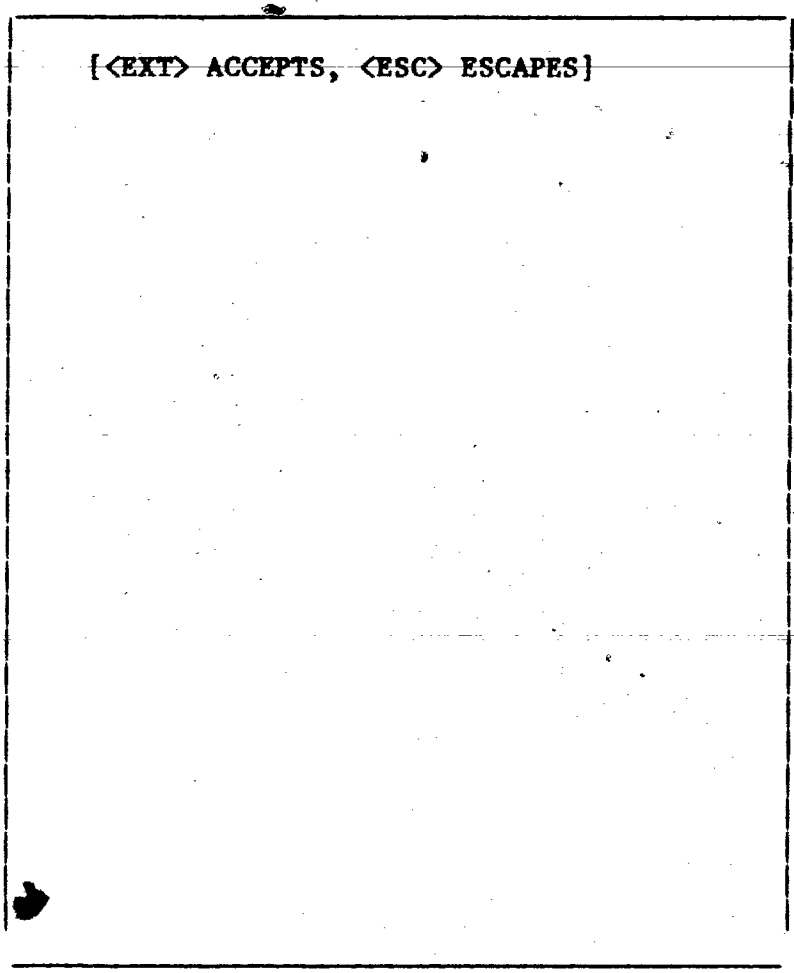


Don't forget to come back to the left hand side of the screen. (Press "CTRL-A").

6. We next want the command I(NSRT to start inserting lines of text into our workfile. Press the key "I" to enter I(NSRT (insert) mode.
On the left hand side we get:



and on the right hand side:



By now you should know how to use the "CTRL-A" feature to switch back and forth between the left and right hand sides of the screen in order to read the complete set of "I(NSRT" commands. These commands allow for the following choices:

a. Press the backspace key to delete a character that is not wanted

b. "DEL" means press "CTRL-X" to delete the entire line you are working on

c. "EXT" means press "CTRL-C" to make the lines you have just typed in a permanent addition to your workfile

d. Press the "ESC" key to get rid of everything you have typed in since your most recent entry into "I(NSRT" mode

The cursor (the white square which indicates where on the screen you are about to work) will be seen at the top left corner of the screen. This is where you will start to type in your PASCAL program under the insert commands.

Now type in the following program. Press the "RETURN" key at the end of each line in order to begin a new line.

Again, remember to start typing under the insert commands. Do not type in the folowing line:

">INSERT: TEXT [<BS> A CHAR,<DEL> A LINE]".

This is an APPLE-PASCAL system prompt, not part of the PASCAL program.

```
>INSERT: TEXT [<BS> A CHAR,<DEL> A LINE]
PROGRAM SUM(INPUT,OUTPUT);

(* THIS PROGRAMS READS IN TWO NUMBERS *)
(* AND PRINTS OUT THEIR SUM           *)

VAR NUM1,NUM2,TOTAL :INTEGER;
BEGIN
    WRITELN('ENTER TWO NUMBERS?');
    READ(NUM1,NUM2);
    TOTAL:=NUM1+NUM2;
    WRITELN('THEIR SUM IS ',TOTAL)
END.
```

7. When you have finished typing the last line press "CTRL-C" in order to enter these lines permanently into your workfile and return to the "E(DIT" level of commands.

8. To leave the E(DIT mode and return to the outermost command level, press "Q" to Q(UIT.

The screen will then show this:

```
>QUIT:
  U(PDATE THE WORKFILE AND LEAVE
  E(XIT WITHOUT UPDATING
  R(ETURN TO THE EDITOR WITHOUT UPDAT
  W(RITE TO A FILE NAME AND RETURN
  S(AVE WITH THE SAME FILE NAME AND RETURN
```

The above prompts allow you to make one of five possible decisions concerning your workfile (Your workfile contains the program you have just typed into the computer.):

a. U(PDATE - keep a permanent record on disk (your APPLE1 or APPLE2 disk) of the work you have just typed in before returning to the outermost command level

b. E(XIT - throw out the work that was just typed in before returning to the outermost command level. No record of the program that was just typed in will be kept

c. R(ETURN - Go back to E(DIT mode. (You may have just remembered something you forgot to type in, there may be some corrections to make, or you may have typed "Q" by accident while still in E(DIT mode.)

d. W(RITE - save the "workfile" as a permanent file under a name of your own choosing and then return to the outermost command level

e. S(AVE - if the "workfile" was a copy of a file already in existence, you may re-save this latest version under its old name. (To understand how this feature works, see the G(ET command in the APPLE-PASCAL Operating System Reference Manual.)

9. In response to the above prompts, press "U" for "U(PDATE". This will cause your workfile to be updated on your disk under the name "SYSTEM.WRK.TEXT" as a permanent record of your program. Since "SYSTEM.WRK.TEXT" will always be the name of any file or program you are currently working on, you may wish to save this program under an other name. At this point it would be a good idea to read about the "S(AVE" command in the APPLE-PASCAL Operating System Reference Manual. Also, find out how to display all your file names.

10. You should now be back at the Outermost command level. Press "R" for "R(UN".
This will set the stage for the PASCAL compiler to process and execute your program. This process takes place in two stages:

a. the "C(OMP" (compile) stage in which your program is checked for syntax (spelling etc.) errors and

b. the "R(UN" stage in which the compiled version is executed, i.e., in the program you just typed in, you should be prompted to "ENTER TWO NUMBERS?". Enter two numbers separated by at least one space.

At this time make sure the sum of the two numbers will not exceed 32767 or -32768. The reason for this restriction will be discussed later on in the course. Do not worry. APPLE-PASCAL allows you to work with larger integer numbers when necessary.

Having entered the two numbers the program will proceed to print out their sum, after which, its work being done, the program will stop.

11. There may however be one slight problem. If you did not type in the program correctly, processing will not pass the compile stage. Any errors detected will be printed on the screen and you will be prompted to either continue, or revert directly back to "E(DIT" mode. If this is the case you should go directly back to "E(DIT" mode and attempt to correct the error. You may not recognize just where the error has occurred. Seek help from your TA or Instructor when necessary. Do not waste time with random guesses.

Once you have detected the error, you will need to make use of several of the "E(DIT" commands which control the screen editor. This means that you make changes to the lines of text directly on the video monitor by first moving the cursor to the place in the text that you wish to change and then using the "E(DIT" commands to A(DJUST, C(PY (copy), D(LETE (delete), I(NSRT (insert), etc.

Do not put off learning how to use the "screen editor". You can become expert at it in a very short while and this knowledge will serve you well throughout this course and into the future.

12. Once your program has compiled and run correctly, please do not forget to remove your disks from the disk drives and turn off both the computer and video monitor.

CHAPTER I

The objective of this self-directed study course is to develop the students' computer programming knowledge to a level such that the student will be able to utilize this facility as a potential tool or aid for problem solving.

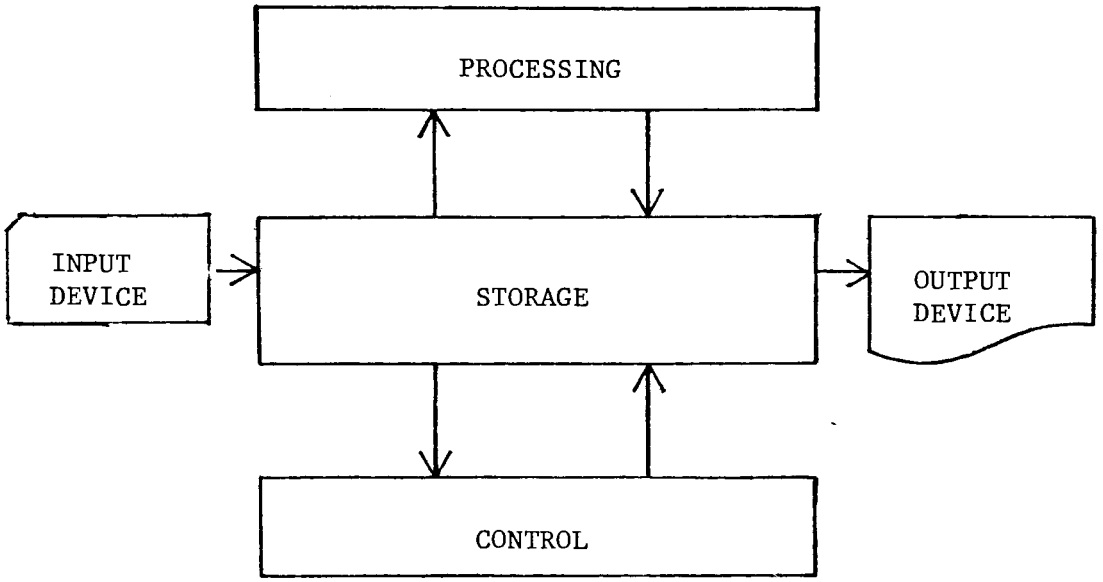
Problem solving is not new. We have all been solving problems for years. Generally, the same basic approach that one would use to solve a problem manually will provide the best computer solution as well. Normally it will require more effort to solve a problem for "one case" using a computer than it would take to do it manually. Hence the effort of programming is only justifiable where considerable repetition in some form or an other is involved.

Our human ability to perform many simple actions in one step is our biggest hazard when trying to create computer solutions for a problem. The computer must be told every last detail of every step. For example, have you ever considered what is involved in the simple action of counting 1,2,3,4,5 etc. First, what is 1,2,3? It is a sum, kept in a particular location of our memory. A sum of what? A digit one, from an other location of our memory, added to the previous sum. But, how did the "process" get started? For us, starting the process is "common sense", we always begin at "one". However a computer has no "common sense". It will begin counting at eight or nine or wherever it is told to start counting. Further, a computer does not automatically have either a "one" or a "place to hold the sum". Both of these must be "requested" along with all the other steps required.

To simplify our communication with computers various "languages" have been developed. The language you will be using is PASCAL. Once a knowledge of the PASCAL language has been acquired, you can 'state' your computer requirements and problem solution steps in PASCAL rather than in your own language. But PASCAL is still not the language the computer understands.

The computer's language is usually referred to as "machine language" or in the case of PASCAL it is called P-Code. Although machine language is not the concern of this self-directed study course, since a special "go-between" called a COMPILER or TRANSLATOR handles this final communication step, some awareness of what the computer "looks like" is worthwhile.

For programming purposes it may be useful to think of the computer as having five basic components:



1. Input Device -

   This is similar to our eyes. It reads and makes data (information) available to our system.

2. Output Device -

   This is similar to our voice or writing. It makes results or answers available to others.

3. Storage

   This is a warehouse for:

   a. input data - each element of information must be put into a separate "box" in storage.

   b. output data - every answer or result must come from a "box" in storage.

   c. intermediate results - all "work in process" must have its own box as well.

4. Processing -

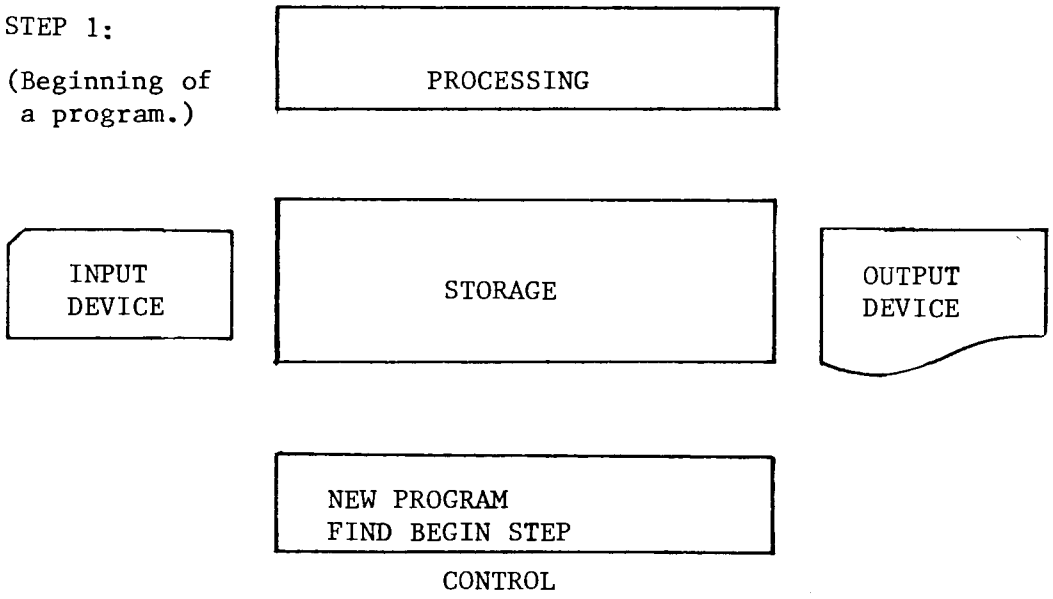
This changes data coming from storage by adding data elements together ,comparing two data elements and noting which one is larger, moving data from one box to an other box, etc. All "results" of the processing are returned to storage again. Usually a new "box" is used but sometimes the new results may be put back into an old "box".

5. Control -

This is the overseer. It keeps track of the program and directs the computer to do the requested steps one after an other.

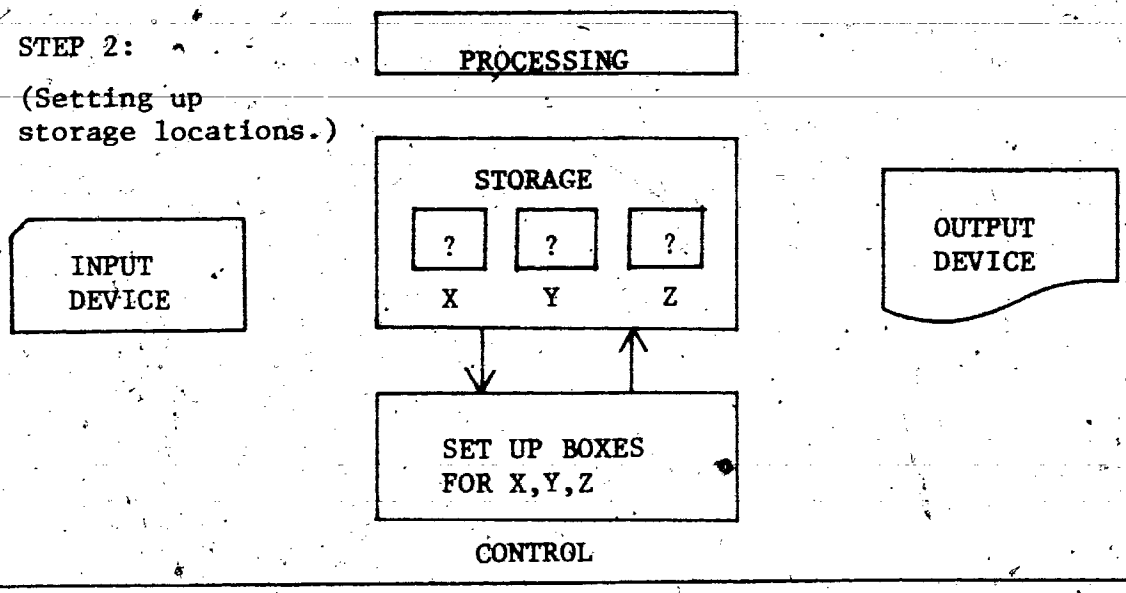
Now, let´s see what happens to our five components when the computer works out the following problem which involves two data elements, referred to as X and Y.
We want the computer to read these two data elements, add them together and print out a result, referred to as Z.

"Control" always takes the initial responsibility. The necessary steps to solve the problem might be:



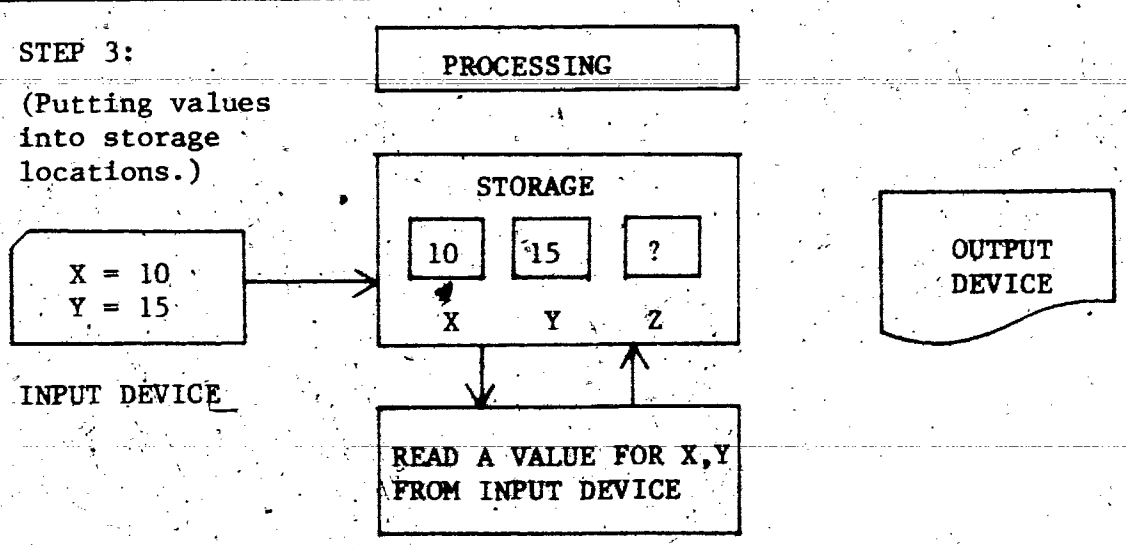
STEP 2:

(Setting up storage locations.)



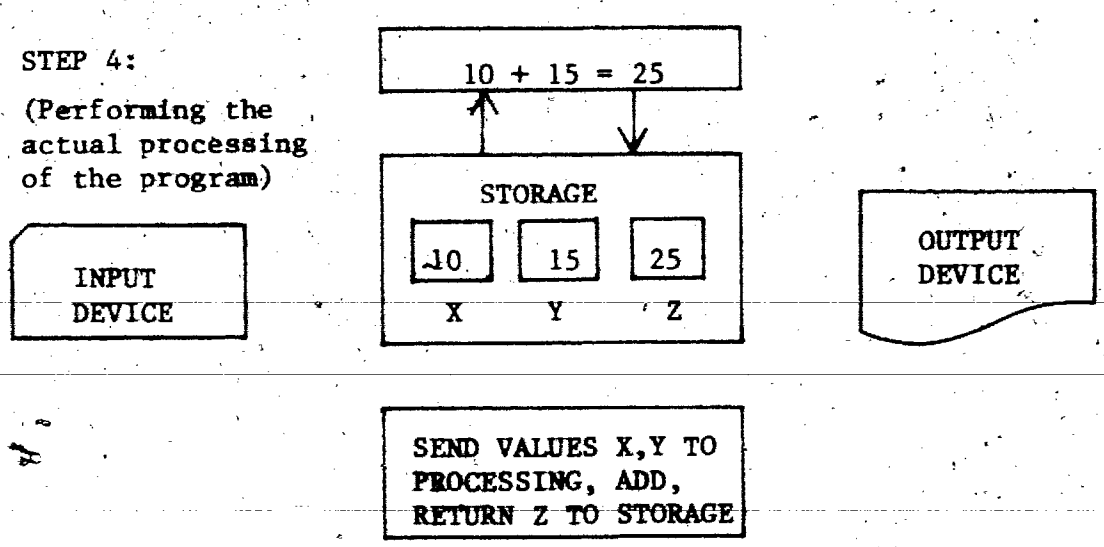
STEP 3:

(Putting values into storage locations.)



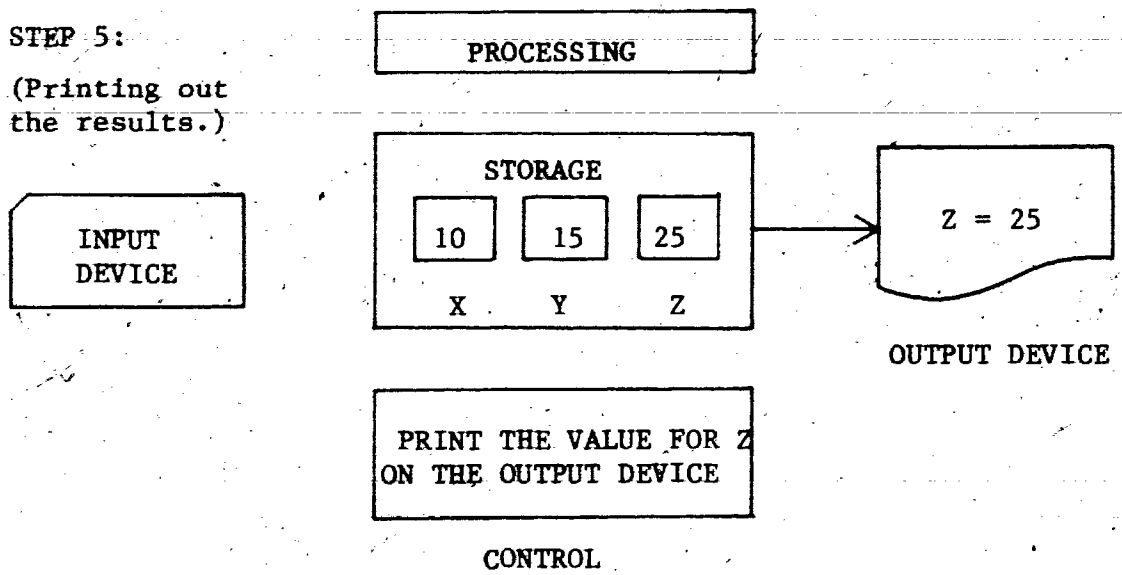
STEP 4:

(Performing the actual processing of the program)



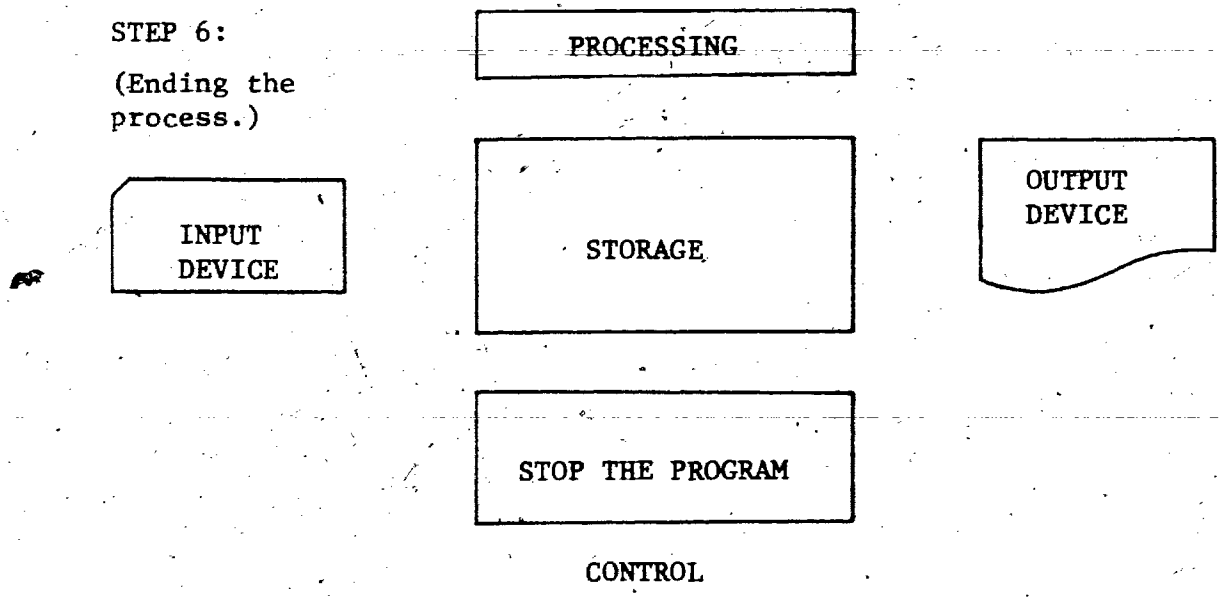
STEP 5:

(Printing out the results.)



STEP 6:

(Ending the process.)



Reminder

During your readings of the text and study guide you may come across certain terms that do not make sense to you. Please come to the Lab. We shall be happy to discuss these terms further and provide answers to anything that may be puzzling you. Come and ask questions. That's what we are here for.

Part A: Read the Preface and chapter 1 of the text.

Pay special attention to the section "Algorithms and Data Structures". It is almost impossible to sit down and code a good solution to a programming problem at a terminal. Some people try it but this approach usually results in wasted time and an inefficient, sloppy solution. And more often than not the solution is incorrect.

Note that we will be using UCSD PASCAL on the APPLE-PASCAL microcomputer system. The UCSD additions greatly enhance the PASCAL language and you will find it worthwhile to incorporate these features into your programming.

READ and WRITE (also READLN and WRITELN) statements are commonly referred as the I/O (Input and Output) statements. All programs must produce output and most programs require some form of input. Can you think of a computer program which doesn't require input?

Note:

The main example (the second one) in this chapter of the textbook is a numeric one. The prime reason being that numeric problems are shorter to describe and shorter to "code": If any of the material presented is confusing, seek help; do not struggle or avoid the issue. Your main emphasis should be on the order (when to do what) and on the use of PASCAL syntax, not on the arithmetic.

Documentation and readability are extremely important in the use of any programming language.
Written comments inform the programmer how, when and where the code relates to each step of the solution algorithm.
Indentation clarifies the code and shows the dependence and independence among sets of programming statements.

Part B: exercises

Note:

Exercises are included after every chapter of the text. Their purpose is two-fold. One to provide a chance to reflect on your reading and two, to provide an opportunity to point out "limits" or "problem areas" for certain instructions. If any of the exercise questions or answers do not make sense, come to the Lab and ask us about them.

Now try the exercises for chapter 1. Check your answers with those at the end of the textbook.

Part C: Reflect section

1. Where in the program would you find the following statements

   a. PROGRAM

   b. VAR

   c. END

2. What PASCAL statement is required to display (print) information on an output device?

3. What PASCAL statement is required to bring information into the computer from an input device?

4. Which of the following are NOT members of PASCAL's permanent vocabulary (keywords)?

PROGRAM

VAR

SUM

READ

A

B

WRITE.

If you cannot answer the above questions you should re-read this chapter and seek help from a T.A. or Instructor.

Make sure that you understand all the concepts of chapter 1 before proceeding to chapter 2. You will often find that when some material on the page you are reading doesn't make sense to you, the reason is that you have missed some of the information on an earlier page.

Part D: Assignment 1

Sign onto the APPLE-PASCAL system and:

a. Type the program on page 8 of the textbook into the computer

b. Compile and Run the program (Both steps can be done by issuing only the R(UN command).

c. Hand in your C(OMPiled version ONLY. In order to obtain a compiled listing of your program, type in the line (*$L*) as the first line of your program. Doing this will produce a file on your APPLE 1: disk called SYSTEM.LST.TEXT. You can then T(ransfer this file to the printer (PRINTER:). Make sure that your apple is connected to a printer and that the printer is turned on before attempting to print out your listing.

No other work, i.e. flowchart, user documentation, etc., need be handed in with this assignment.

Please hand in your assignment ONLY during the scheduled lab. periods.

Part E: Flowcharts

On page 2 of the textbook , you are introduced to the concept of an "algorithm" - a set of steps which must be followed sequentially in order to solve a problem. The examples of written solutions which follow in the text are straight forward and unambiguous. However, as problems get more complex, the written solution tends to become less clear and more open to ambiguities. The clarity of the written solution is completely dependent upon the logical way in which you present your algorithm.

Program development is greatly improved when the solution algorithm can be expressed clearly and unambiguously. One method of achieving this is through an illustrative diagram called a flowchart.

The flowchart, a series of box-like symbols, charts the flow of the reasoning you followed in solving the problem. The uses of some of the different shapes are shown on the following pages of the study guide. Each box contains an instruction or a set of instructions in a natural language, in this case English, and represents a single step of the solution algorithm. The lines connecting the boxes have directional arrows showing the order in which instructions are to be carried out next.

Hence there is never any doubt as to what the steps of the solution algorithm are, or the order in which they are to be implemented. Control of the algorithm flows from one box to the next until the solution is complete.

Since flowcharts are unambiguous, they are a valuable tool in communicating an algorithm from one person to another, eg., from the problem solver to the computer programmer. (On large programming projects these jobs are usually performed by different persons.) If the flowchart is very detailed your PASCAL code can be derived almost directly from the boxes.

Finally, the flowchart plays an important part in the documentation of all computer programs. When programs have to be changed or modified, it is often easier to understand the logic of a program by referring to the flowchart rather than to the actual program.

Some textbooks present a hierarchical approach to flowchart design, what is technically called a "hierarchy flowchart". In a hierarchy flowchart many steps are condensed into one single box.

As the emphasis in this course is on detail design, we prefer you to use the detailed flowcharting methods set out in the study guide. Flowcharting will be discussed more fully in the course lectures.

There are many methods of detail flowcharting other than the one presented here. If you wish to use one of these alternative methods on your assignments, please consult your T.A. or Instructor before handing in your work.

WORDING

Try to use everyday language whenever possible:
e.g.

Read A,B,C or GET A,B,C	not	READ(A,B,C) or READLN(A,B,C)
Reserve storage for X, Y	not	VAR X,Y :INTEGER
A=B	not	A:=B;

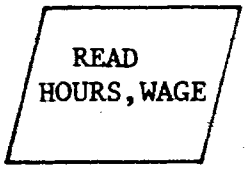
Certain PASCAL type statements have been accepted as "abbreviated English". These may be used on the flowchart.

A=B	or	A <- B
SUM=SUM+1	or	SUM <- SUM+1

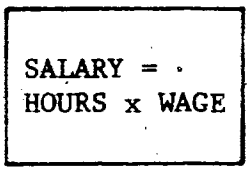
Flowcharting Symbols

NOTE: Use flowcharting templates available in the bookstore. They will save you hours of time.

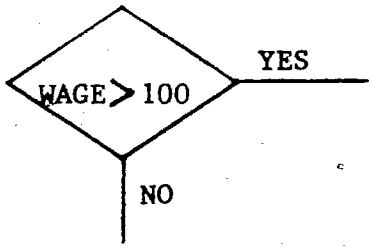
START and STOP



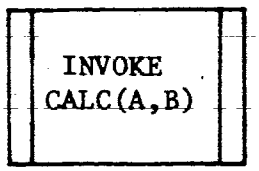
INPUT and OUTPUT
(I/O)



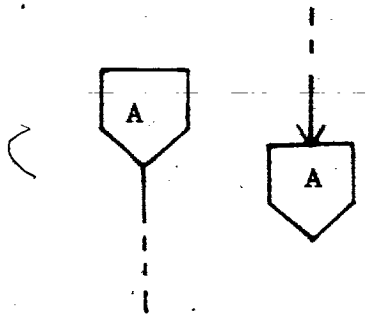
PROCESS



DECISION

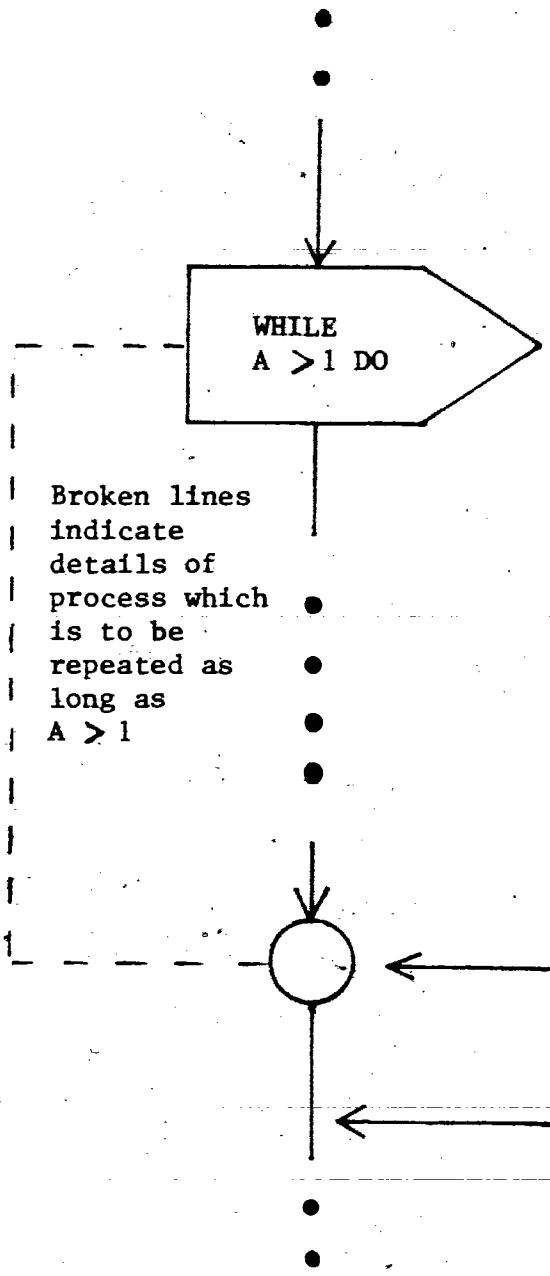


PROCESSING MODULE INVOCATION
(Invocation of a SUBROUTINE or FUNCTION)



Offpage Connector

(Onpage Connectors exist but should never be used in a structured flowchart)

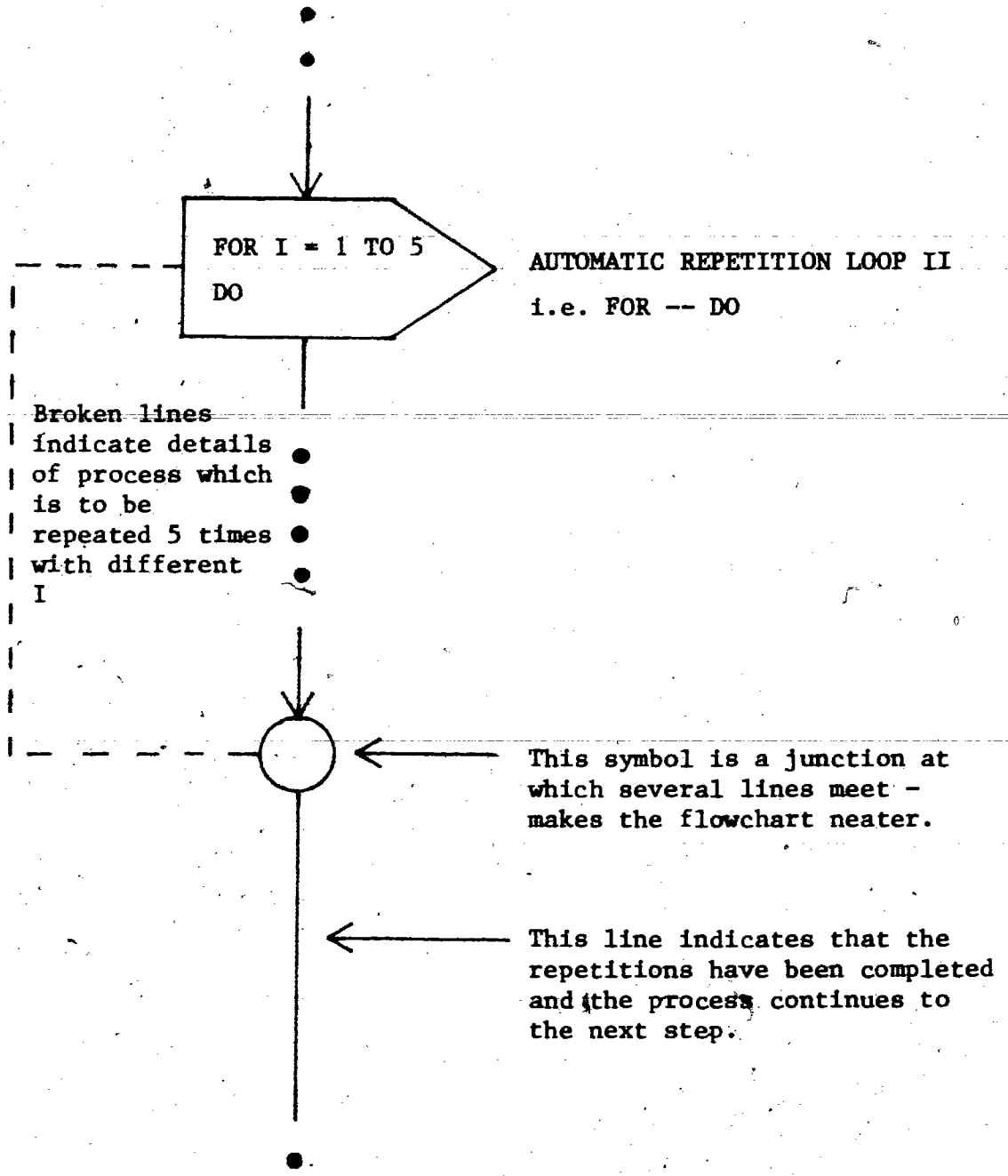


AUTOMATIC REPETITION LOOP I

i.e. WHILE -- DO

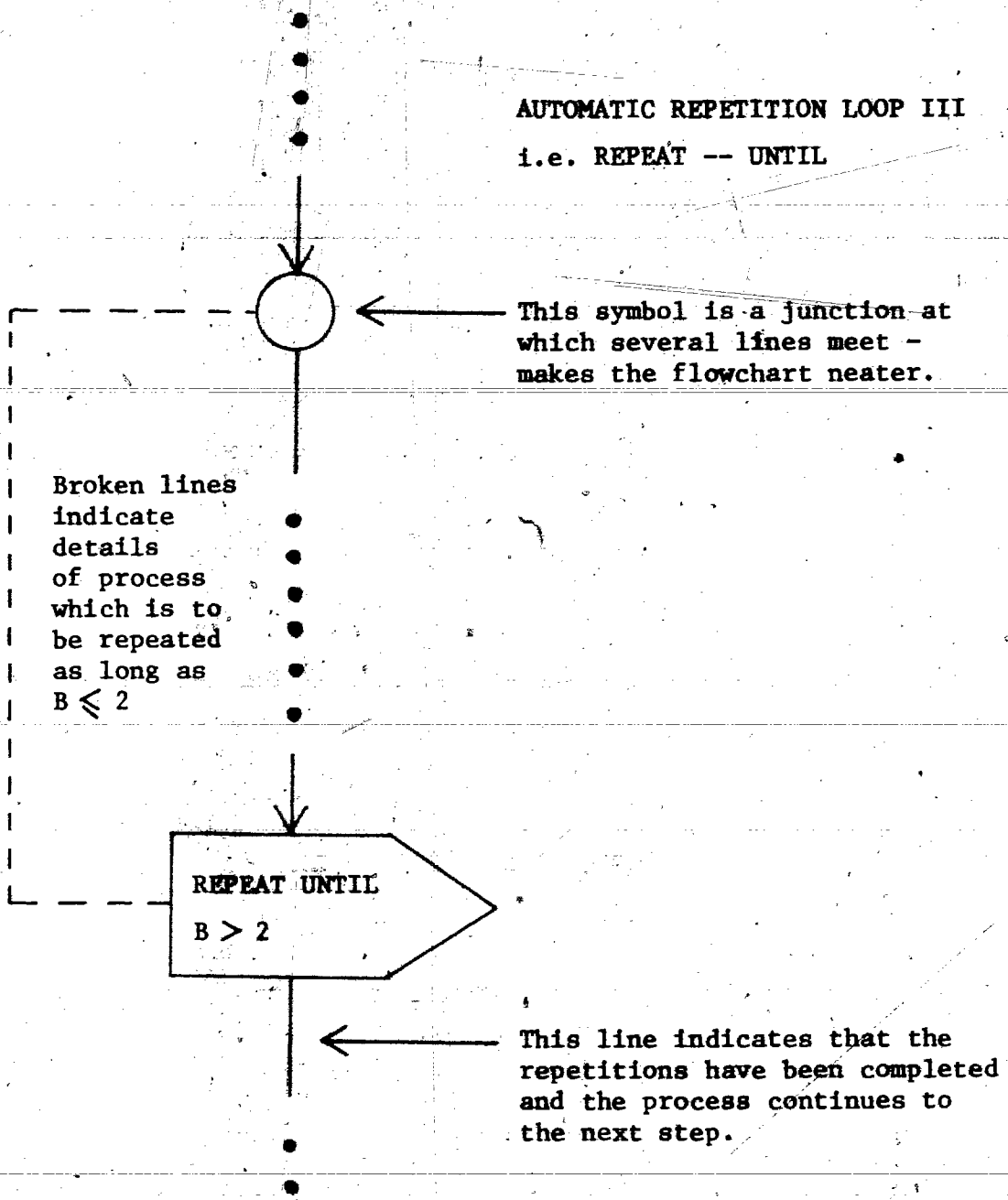
This symbol is a junction at which several lines meet - makes the flowchart neater.

This line indicates that the repetitions have been completed and the process continues to the next step.


FOR I = 1 TO 5
DO
AUTOMATIC REPETITION LOOP II
i.e. FOR -- DO
Broken lines indicate details of process which is to be repeated 5 times with different I
This symbol is a junction at which several lines meet - makes the flowchart neater.
This line indicates that the repetitions have been completed and the process continues to the next step.

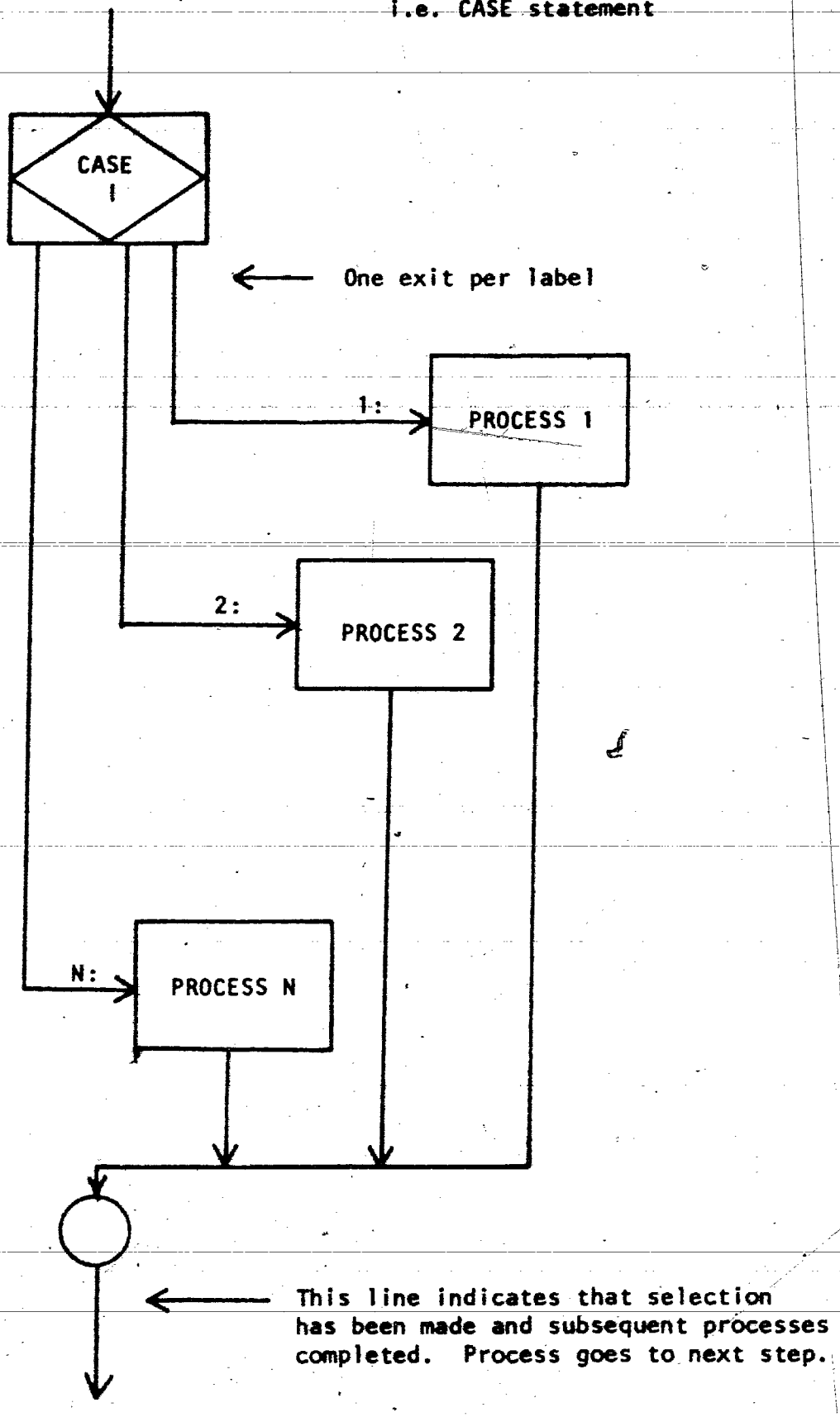
AUTOMATIC REPETITION LOOP III

i.e. REPEAT -- UNTIL

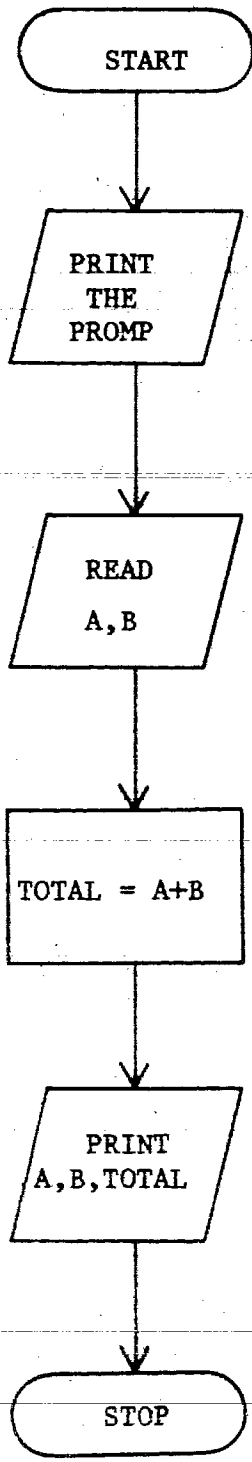


Selection statement:

i.e. CASE statement



Flowchart of Assignment 1



d. BOOLEAN - the representation of TRUE and FALSE conditions.
All computers function on the basis of certain conditions being either TRUE or FALSE. This is also true in most programming situations. Setting an INTEGER variable to 1 or 0 to handle a TRUE or FALSE situation would produce the same results as the BOOLEAN TRUE or FALSE. However the BOOLEAN approach uses only one eighth of the storage requirement.

2. -MAXINT and +MAXINT are useful for file manipulation in a data base environment. Do not worry about their application at this point.

3. The remainder left after a division can be obtained with the MOD function. Since division is usually performed to obtain a quotient, the usefulness of the remainder is not readily apparent, but consider the following:
Suppose with a budget of $12000, we wanted to buy computer terminals at $1073 each. If we used our entire budget to buy these terminals, how much money would be left?

Solution 1:

Keep subtracting 1073 from the original 12000 until the amount left is between 0 and 1072

Solution 2:

Use the MOD fuction to find out how much remains after dividing 12000 by 1073

Another useful application of the MOD function is the generation of random numbers. This technique will be left to a future exercise.

4. The ABS (absolute value) function is used to find an absolute difference between two values where the positive or negative aspect of the number is irrelevant.
For example:
Suppose we need a program that will take as input the hockey scores of two teams and print out the point spread between the winner and the loser. By using the ABS function there is no need to worry about the sequence in which the team scores are input into the program (winner followed by loser or visa versa). Whether the subtraction produces a positive or negative result, the ABS fuction will always give a positive result.

e. The wording should be kept brief and in everyday language avoiding the use of PASCAL language if possible.

f. The flowchart should be easy to understand.

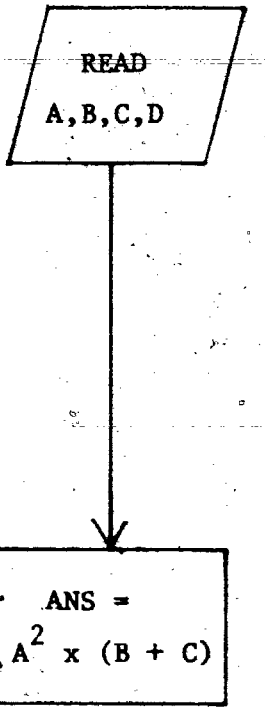
g. It should show all logic and input/output steps separately.

h. "Automatic Repetition" constructions should have dotted lines without arrows.

LEVEL OF DETAIL

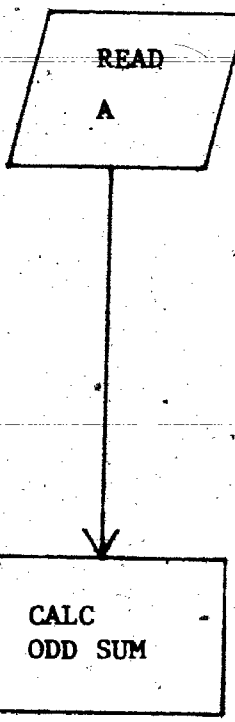
Use separate blocks for each type of activity such as reading, calculating, printing and logic.
Similar statements may be grouped if the time sequence is the same, if no logic decisions are involved and if a suitable description is given.

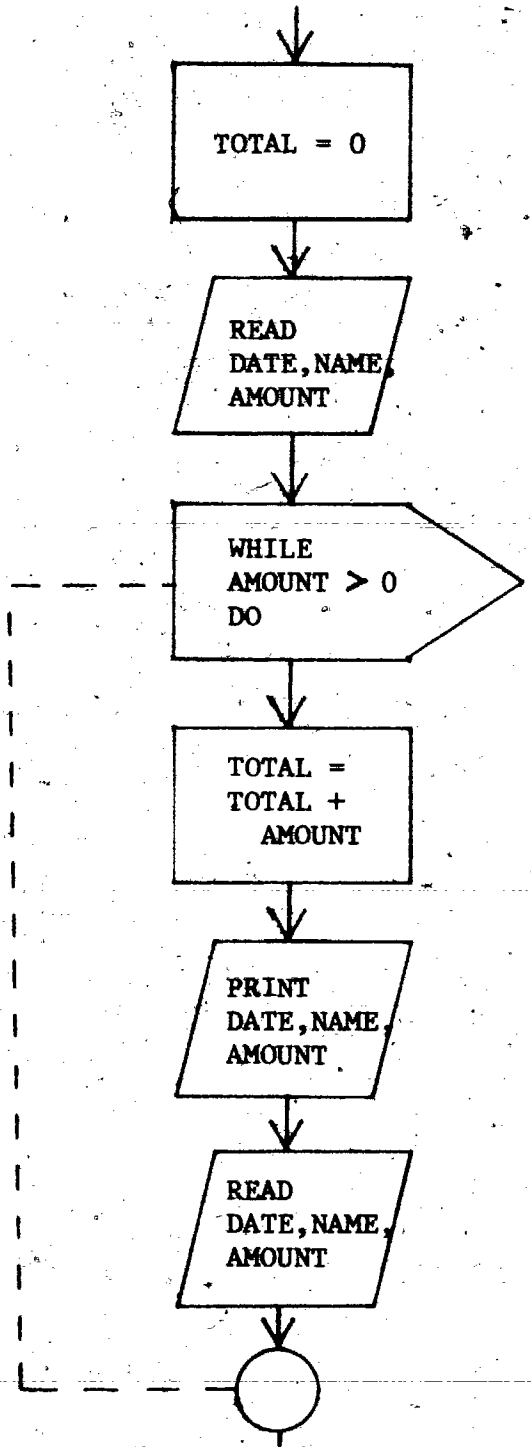
OKAY

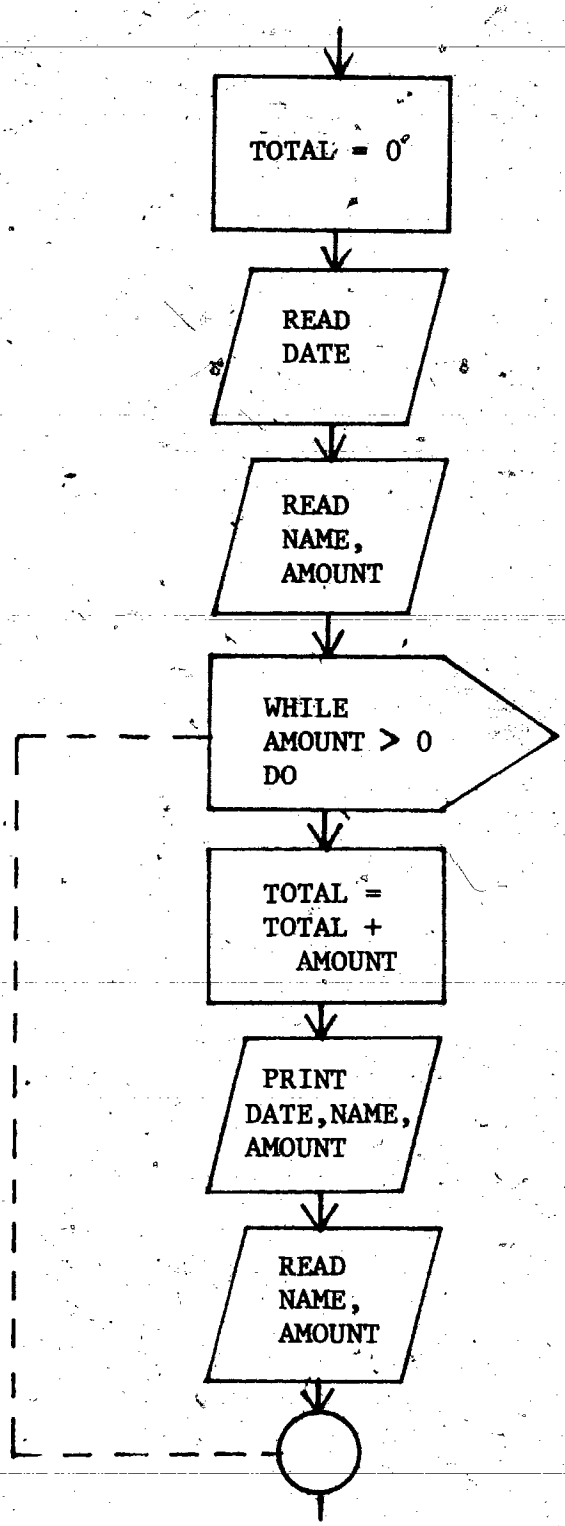


A,B,C,D and ANS are variable names used in the program.

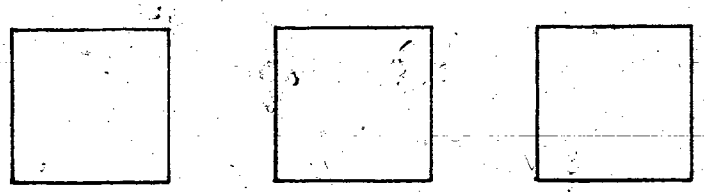
NO -- The logic required to calculate ODD SUM must be spelled out



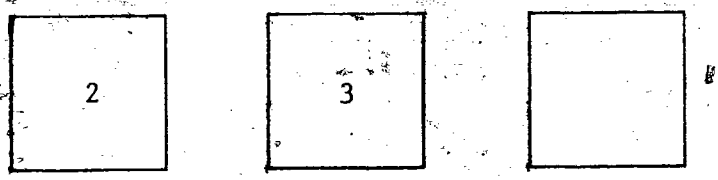




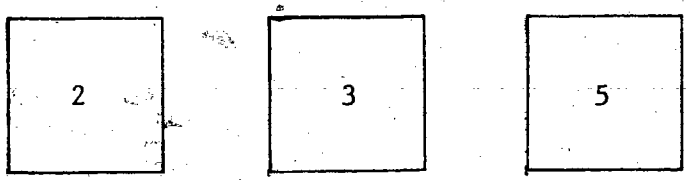
The input data requirements are very different for the two flowcharts above - make sure your flowchart suits your data.



At the beginning of the program

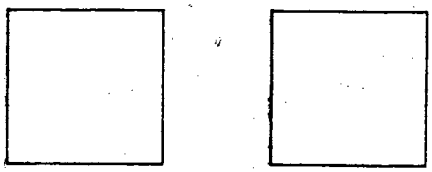


After the READ (A,B) statement

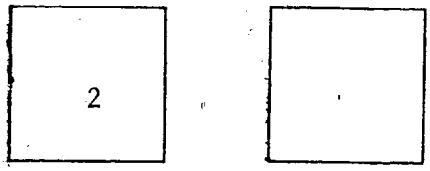


After the TOTAL:= A+B statement

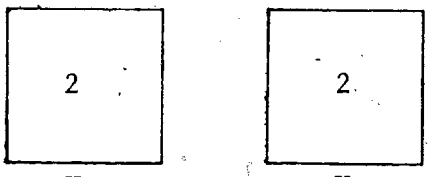
Figure I.A



Before execution of a program in which storage for variables named X and Y were reserved.



After the statement X: = 2



After the statement Y: = X

Figure I.B

CHAPTER II

Having considered how a computer solves a problem, let's now look at what actually happens when we solve a problem on a computer.

First everything our computer needs to know must eventually be in a form appropriate for that computer. As you already know, the computer system for this course is the APPLE and the language is PASCAL.

Four different stages are involved in the "solving" of a problem:

1. planning the solution (preparation of the solution algorithm)

2. preparing the code and typing it into the computer

3. compiling the program. At this stage the program is checked for correct syntax. This means that the compiler checks to see that you followed all the rules governing format and language and that you did not make any spelling mistakes. If the syntax is correct, the program will be translated into P-Code

4. running the program (sometimes called "executing the program"). This is the stage at which the P-Code instructions (the compiled version of the PASCAL instructions) are carried out by the computer

You should by now be familiar with all four of these stages.. If you aren't, ask.

Part A:

Read chapter 2 of "Introduction to PASCAL Including UCSD PASCAL".

Pay special attention to "FORMAL ORGANIZATION OF A PROGRAM". Even though the definitions and declarations are optional, when are used, they should be in the exact order laid out in the textbook.

Most important of all you must learn to recognize the "reserved words". This will be easier to do as you acquire more experience with the PASCAL language. The illegal use of reserved words can cause many hours of frustration to the programmer when trying to find the error(s) in a program that is otherwise correct.

The compiler program does more for us than just translate "our program" into machine language form. It can also (on

request) produce a "listing file".

The listing file resembles our input program; most of it is actually an exact copy of our program lines. However, the most important feature of the listing file is the inclusion of error messages for any incorrect code. This feature gives you the advantage of allowing the compiler to check through your complete program and list all the error messages which can then be printed on paper (by T(RANSFERring SYSTEM.LST.TEXT to PRINTER:). You are thus saved the inconvenience of ending the compile prematurely and returning to the editor to correct one or two errors at a time.

Five columns of information have been added to the left. The first column identifies each line of PASCAL code with a statement number. The next four columns do not contain information necessary to the beginning programer. Those interested should consult the "compiler options" section of the APPLE-PASCAL Operating System Reference Manual.

A listing file may be requested by adding the compiler option "L" before the "PROGRAM" header. Compiler options are specified as a special form of a PASCAL comment. Instead of (* comment *), a $ followed by the compiler option, is placed directly after the first asterisk - no blank spaces are allowed in between.

EXAMPLE:

```
(*$L*)
PROGRAM SUM(INPUT,OUTPUT);
VAR NUM1,NUM2,TOTAL :INTEGER
          .
          .
          .
```

The listing file that will be produced will be called "SYSTEM.LST.TEXT".

Possible Coding Errors

Sometimes a PASCAL compiler error-message may be caused by an error in a earlier statement. Always check for the following obvious errors:

1) missing or badly placed semicolons

2) missing comment indicators (*....*)

3) missing parentheses

4) spelling problems

5) unclosed quotes (Quotes must be closed at the end of every line.)

6) incorrect use of blanks or spaces

Look at the line for which the error message occurs and then check the preceding and following statements as they could be causing the error.

Although it is not necessary to memorize format and syntax rules, careful attention should be given to "detail" when coding. Any uncertainties should be checked, this means knowing where to find the rules.

Part B:

Do the exercises for chapter 2 of the text. They are quite simple and will test your understanding of the concepts presented so far.

Part C: Reflect Section

Check appendix B of the study guide again for the way to prepare and submit your assignments.

Part D: Assignment 2

The purpose of assignments 1 and 2 is to give you practice in using the APPLE-PASCAL system. Complete the work requested below and hand in your Listing file along with your directory listings and answered questions.

1. G(ET your assignment file into your "workfile" if it is not there already.

2. On the first line of the PROGRAM, remove "(INPUT,OUTPUT)".

3. Change the PROGRAM name "SUM" to "SUMUP".

4. After first line of the program add a PASCAL comment giving the current date, eg. (*Sept.21,1995*)

5. Modify the program to find the sum of three numbers instead of two.

6. Add sufficient comments at appropriate intervals to describe the program and the work being done.

7. RUN the program, and print out the "listing file" on the printer.

8. Save your file under the name "ASSIGN2". (Remember, the suffix ".TEXT" is added by the system automatically.)

9. Print out an E(XTENDED directory listing of both your disks. To accomplish this step you will need to become familiar with the T(RANSFER and E(XTENDED commands of the F(ILER. See the APPLE-PASCAL Operating System Reference Manual.

10. Use your own words to explain the following in detail.

a. How to display all those F(ILER options which do not appear on the screen's two 40-character "pages".

b. How to make a backup copy of your disk.

c. How to erase a file from you disk.

d. How to combine unused blocks on the disk.

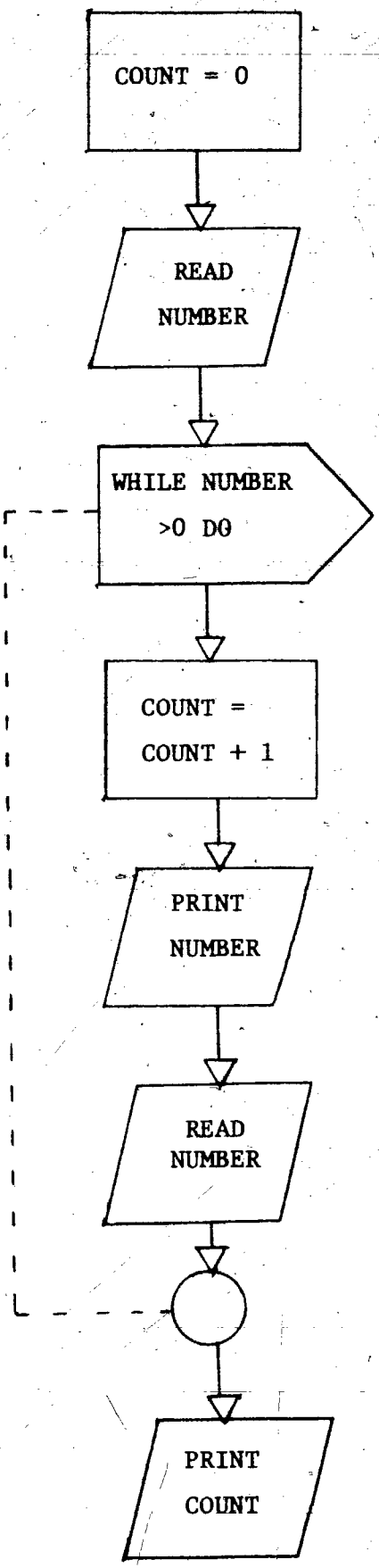
e. Why removing "(INPUT,OUTPUT)" in step 2 above will not cause an error.

Note:

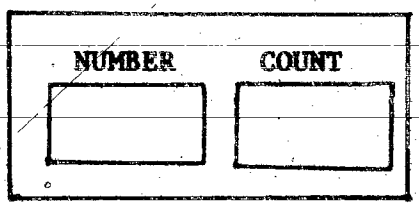
User documentation and flowchart which are the same as the example assignment in APPENDIX B of the study guide, need not be handed in.

PLEASE HAND IN YOUR ASSIGNMENT DURING SCHEDULED LAB. PERIODS ONLY.

Remember to hand in your LISTING FILE.

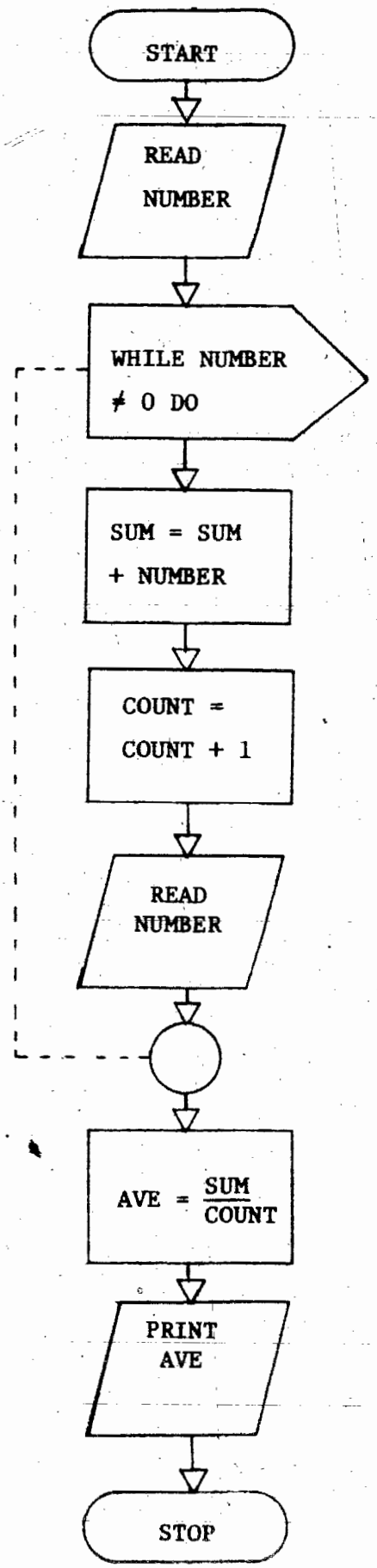


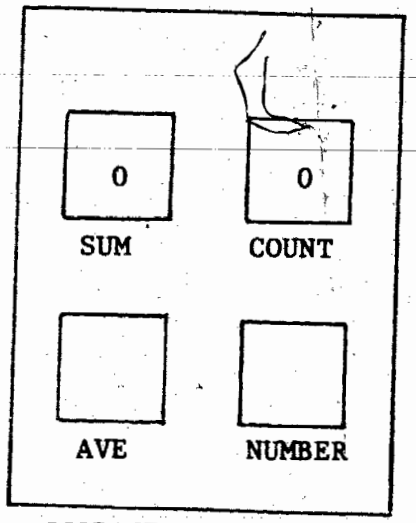

```
COUNT: = 0;
READ (NUMBER);
WHILE NUMBER  >0  DO
    BEGIN
        COUNT: = COUNT + 1;
        WRITELN (NUMBER);
        READ (NUMBER)
    END;
WRITELN (COUNT);
```



STORAGE LAYOUT

FIGURE II.A





STORAGE LAYOUT

FIGURE II.B

CHAPTER III

Although there are no specific assignments related to chapters 3 to 5 of the textbook, it is vital that you understand the material presented in them. Once you understand this material, and you must, you will be able to recognize the situations in which these features of the PASCAL language should be used.

Throughout this course you will find the need to refer to the information in these chapters in order to find the best methods of representing your solution algorithm as PASCAL code. Keep in mind that in computing there are many ways of solving any given problem and arriving at a correct solution. Being correct is not enough. Your solution must also be efficient.

Part A: Read chapters 3, 4 and 5 of the Textbook

The following is a supplementary explanation of the more important concepts presented in chapters 3, 4 and 5:

1. All computers work with and act upon data. Data are mostly in the form of numbers, and letters of the alphabet. Both these forms of data can be broken down further into more basic data types. For example numbers can be described as whole numbers (integers) or fractions. In PASCAL "letters" may be either single characters or groups of alphanumeric symbols (numbers and letters combined).

   Computers function best when they can deal with each classification of data separately. In other words a data type is a group of data with characteristics in common.

   PASCAL has four basic data types.

   a. INTEGER - these are the whole numbers only.

   b. REAL - any mixed number or fraction.

   c. CHARacter - a single alphanumeric symbol. In PASCAL CHARacters are always enclosed in single quotes and may consist of a letter of the alphabet, a single digit, or any other "legal" symbol (i.e. $,#,&) belonging to the PASCAL language.
   Note that a digit in quotes is considered to be a CHARacter and its internal representation inside the computer is different from the INTEGER representation of the same digit.

Be sure to draw a storage layout with your flowchart (details at the beginning of this chapter). The starting value of any initialized variable should be shown in the storage layout. (Put the correct value inside the box.) Notice there are three distinct timings for this program, the "Initialization" (or beginning), the "main loop" and the "ending" where the final averages are printed out.

NOTE: FOR THIS ASSIGNMENT AND ALL FUTURE ASSIGNMENTS WHERE SAMPLE INPUT AND SAMPLE OUTPUT DATA IS PROVIDED, YOUR INPUT AND OUTPUT WILL NOT APPEAR ON YOUR TERMINAL IN THE SAME FORMAT THAT IT IS PRESENTED IN THE STUDY GUIDE UNLESS YOU:

a. read you input from the KEYBOARD (consult the APPLE-PASCAL Language Reference Manual)

b. read the input from a disk file (Disk files may be covered in this course at the discretion of the instructor.).

Otherwise your program input on output will be displayed on alternate lines. Consult your instructor regarding the format required for each assignment.

Assignment 6

Character data play an important part in many computer problems. Notice that even number symbols are part of the character set. If numbers are not participating in any arithmetic operations it is often better to describe them in characters.

Referring to the flowchart and storage layout at the end of this chapter, code and run the following problem.

Write a program that reads in four words at a time, each word being no longer than 10 characters; the program is to print out each set of words on a separate line, in alphabetical order. After the last set of words has been processed the program is to print out:

a. the number of sets processed

b. the number of sets that were in alphabetical order to start with, and needed no rearranging.

Use the value "STOP" for WORD1 to signal the end of data. The line containing this value should not be treated as valid input data.

5. The CHARacter functions ORD, CHR, PRED and SUCC are useful for comparing ranges of characters as represented by the internal code of the computer. For the APPLE the internal code is called ASCII code.

This comparison feature will become more important to you as you learn how to expand PASCAL's four basic data types. In the meantime you might try to make a mental note of their existence.

****** NOTE THE ERROR ON PAGE 39 ******
(sixth line from the top)
"SUCC(C) = CHR(ORD(C) - 1)" should be "SUCC(C) = CHR(ORD(C) + 1)".

6. There is no substitute for the truth tables presented on pages 40-41. Do not memorize them but make sure you understand each relationship. Get help if you don't!

7. The computer has rules of "operator precedence" which may appear ambiguous to the human programmer. To make sure that your arithmetic expressions are always correct, use parentheses.

8. The compound statement solves the problem of using many statements where only one statement is allowed. The "many statements" are disguised to look like a single statement by enclosing them between a BEGIN and an END.

9. EOF and EOLN are indispensable when the input data for a program are in a "text file" that resides on a disk. These functions do not work in the same manner when input is done directly at the terminal.

When working with a text processing problem (analyzing written material) the data are usually organized into lines, sentences, etc. EOLN will detect the end of each line of input data.
Since the amount of input data is usually not known, EOF will detect the end of the input data.

Part B:

The following exercises are not long and are worth trying:

3-1 to 3-11
4-2 to 4-6
5-2 and 5-7

Part D: Assignment 3

For this assignment you will first have to become familiar with one more PASCAL construct, the WHILE -- DO loop. Read pages 79 and 80 of chapter 6 of the textbook. The concept is very simple. Every statement found inside a WHILE -- DO loop is repeated over and over again until the condition governing the loop is no longer true. If the condition was never true the loop is never entered. You will find the flowchart representation of the WHILE -- DO loop in chapter I of the study guide.

Using the algorithm (in flowchart form) presented on the next page, flowchart and code a PASCAL program that will continually read a single integer from the keyboard, calculate its square, cube, and square root, and print the results on the video screen. The results should be printed in four neat columns, one each for the number, its square, cube, and square root respectively.

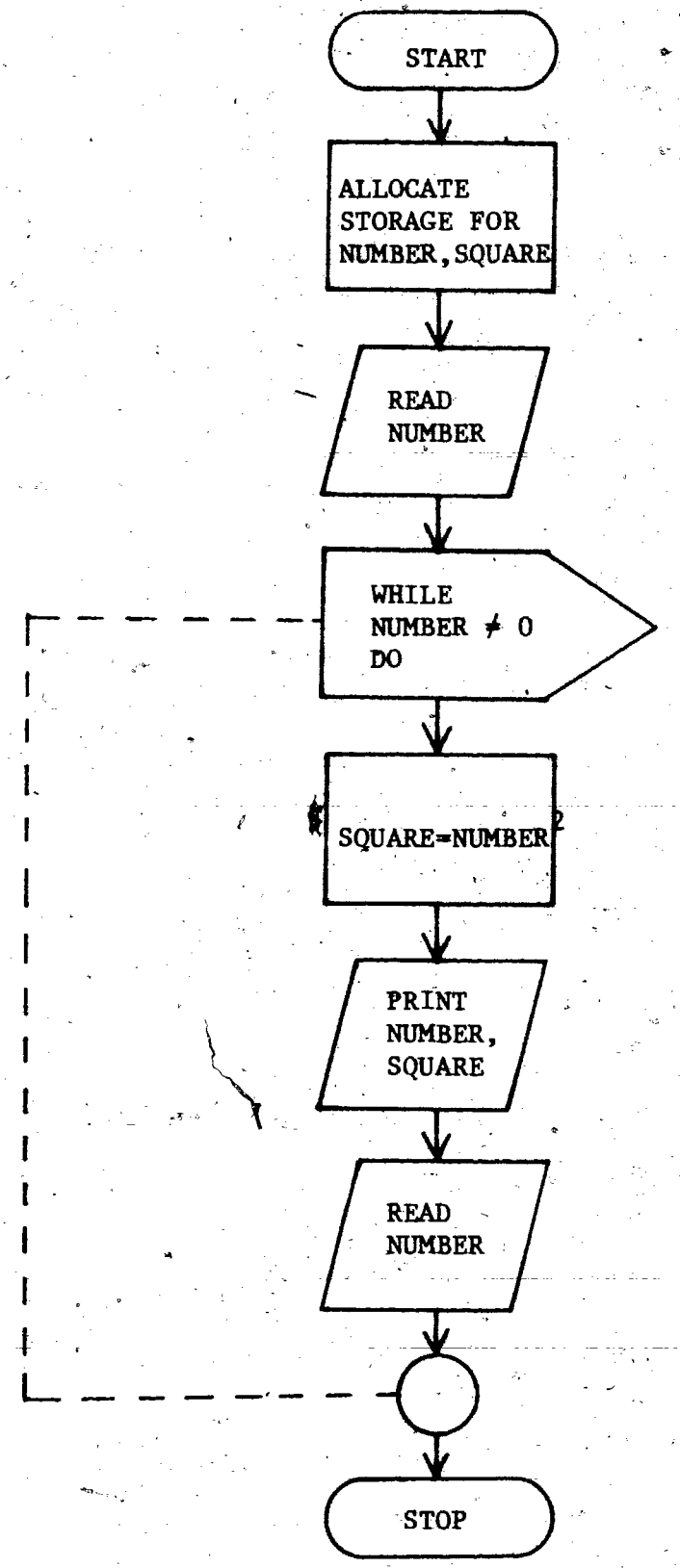
The headings NUMBER, SQUARE, CUBE and SQUARE ROOT should be printed above their respective columns.

Use a "dummy" value of 0 to end the program - this means that the 0 will signal the end of the program but no calculations will be done and no results will be printed for this value.

NOTE: You must add the statement USES TRANSCEND; in order to use the built-in mathematical functions. This statement should be placed directily under your PROGRAM statement.

This assignment and all future assignments should be handed in with full documentation as outlined in appendix B of the study guide.

Flowchart of a program that continually reads in a single number, calculates its square and prints out the results.



CHAPTER IV

Error messages generated by the compiler are of considerable assistance when it comes to getting a program working but remember they can also be a great waste of your time.

By far the most difficult problems to locate are logic bugs because they are not errors of syntax but are errors in timing (sequence). All the error messages in the world can never make a poorly planned program work. Thinking out your program and planning it are essential.

Problem Solving

Planning a program is essentially an exercise in problem solving. The first step is the definition of the problem. The definition can be completely verbal or it can be in the form of a flowchart, or any other method of stating an algorithm. Without such a definition no compiler or list of error messages can help you.

Steps in Problem Solving

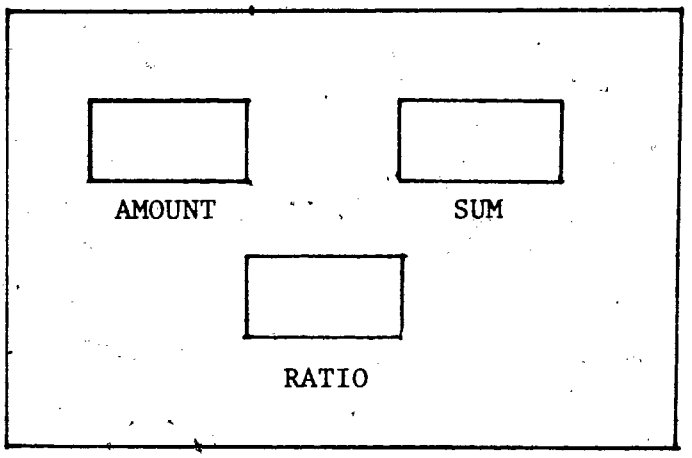
1. The first step in problem solving is a careful reading of the problem and formulating a concise definition of it.

2. The second step is to re-read the definition, review the input, and ask questions about any ambiguities or unknowns.

3. Next make a "stab" at flowcharting a solution; this is best done by selecting some sample input (test data) and following the steps that would be necessary to solve the problem without a computer.

4. Each step will most likely mean one flowchart box. Flowcharts are often best started from the middle, "the key part of the problem", and worked logically from there in either direction.

5. All data required may be shown as "boxes" in storage called a STORAGE LAYOUT which provides a basis for coding the VAR statement later, e.g.



STORAGE LAYOUT

(NOTE:
The value of all variables such as counters which are initiallized at the very beginning of the program may be shown in the storage layout instead of in the body of the flowchart.)

6. Your flowchart should then be reviewed, making sure it will handle all types of input.
Make sure your flowchart is correct before you start coding.

If you are not confident that your flowchart will work, don't go on! All you will do is waste time. Seek help.

Part A:

Now let's take a look at how to approach program development. Read chapter 6 of "Introduction to PASCAL Including UCSD PASCAL".

Pay special attention to the following constructs:

a. WHILE — DO and REPEAT — UNTIL loops

b. FOR -- DO loops

c. IF — THEN -- ELSE statements

d. CASE — OF — END statements

Make sure you understand how they work. You will need to use these contructs in your assignments.

Today's trend in computer programming overwhelmingly favours a top-down structured approach in program design. Modular constructs as opposed to indiscriminate branching are essential.

Those students who have previous programming experience should note that the use of GOTO statements are frowned upon in a structured programming environment. Please do not use GOTO statements in your assignments.
The main purpose of the REPEAT, WHILE, IF-THEN-ELSE, and CASE constructs are to do away with the need to use a GOTO statement anywhere in your program.

NOTE that the textbook uses a slightly different flowchart convention from that of this study guide. The representation of the REPEAT -- UNTIL and the WHILE -- DO statements in the textbook flowcharts could imply the use of GOTO statements in the PASCAL implementation. The structured flowchart convention in this study guide eliminates this type of ambiguity. Please use it.

You should also read chapter 15 before doing the exercises and designing the algorithm for your assignment. This chapter contains some good tips which will help you save time in implementing your programs.
If you find you are having difficulty using structured programming techniques, consult your instructor or T.A.

Part B:

Exercises 1 to 11 are strongly recommended as practice items.
You should have a clear idea of the solution algorithms for exercises 12 to 18 by the time you complete the assignments at the end of this chapter.

Even though you don't do all the exercises, you should study all the answers in appendix L of the textbook.

Part C: Reflect Section

Below, are some questions to reflect upon. If you are not sure of the answers do not hesitate to come to the Lab or ask your T.A. or Instructor.

1. What is the main difference between a WHILE -- DO and a REPEAT -- UNTIL loop?

2. Can a FOR -- DO loop be substituted for a WHILE -- DO loop?

3. Can a WHILE -- DO loop be substituted for a FOR -- DO loop?

4. Why do we need a case statement when the IF -- THEN -- ELSE statement is avialable?

5. Referring to the following statements:

```
IF A > B THEN
   A := A * 2
ELSE
   B := B * 2;
WRITELN(A,B);
```

show what will be printed by the put statement if:

a) A = 75, B = 25
b) A = 15, B = 20
c) A = 25, B = 25

6. a) Will the following routines produce the same output?

```
i) IF TYPE = 1 THEN
       BEGIN
         COST := QTY*(PRICE*.8);
         DISCOUNT := 20
       END
   ELSE
       IF TYPE = 2 THEN
         BEGIN
           COST := QTY*(PRICE*.9);
           DISCOUNT := 10
         END
   ELSE
         COST := QTY*PRICE;
  WRITELN(COST,DISCOUNT);


ii) IF TYPE = 1 THEN
      BEGIN
        COST := QTY*(PRICE*.8);
        DISCOUNT := 20
      END;
    IF TYPE = 2 THEN
      BEGIN
        COST := QTY*(PRICE*.9);
        DISCOUNT := 10
      END;
    IF TYPE = 3 THEN
      COST := QTY*PRICE;
    WRITELN(COST, DISCOUNT);
```

b) What is the difference between routine i) and ii)?
c) What will happen in each routine if TYPE = 4?
d) How would you recommend handling types other than 1,2 or 3?

Part D: Assignment 4

Flowchart and code a program that will input any purchase price and give change from a $1.00 bill. Change should be made in quarters, dimes, nickels, and cents.

An appropriate error message should be printed on the video screen if the amount entered is less than one cent or greater than one dollar. Verification of the input data is known as an error check.
Error checks on the input data are an integral part of every good program.

The program should continue processing new input data until a value of -99 is read in as a purchase price.

Your detailed flowchart should be similar to that discussed in chapter I of the study guide. A good flowchart should have only one start box and one stop box.
Try to implement a good top-down design.

Remember to hand in your assignment with full User and Programmer Documentation. along with your flowchart and PASCAL program.

Part E: Character Variables

Not all computer work is accomplished by arithmetic computation. In many types of data there is a need to represent names. In this chapter we will also take a brief look at CHARACTER variables and their implementation.

You already know about the CHAR data type. Variables given the CHAR data type may contain any one symbol available on the installation (in our case the APPLE-PASCAL system).

What if we wanted a variable to hold character symbols, for example, names like JOHN or SUSAN? UCSD PASCAL (this is the version of PASCAL you are using on the APPLE) allows us to solve this problem by creating a structured data type called the STRING data type. The declaration

```
VAR NAME :STRING
```

would give you a variable called NAME of the data type STRING. You could now place a name of up to 80 characters long inside NAME by simply coding

```
NAME:='JOHN'
```

You may also read in a value for NAME with a READ or READLN statement, i.e., READ(NAME).

NOTE: string data in PASCAL code must always be enclosed in single quotes. This does not apply to string data that is read into the program.

Note also that the quotes themselves are not stored inside the storage location with the data. The reason for having quotes in the first place is to differentiate between string data and variable names, i.e.,

```
VAR NAME          :STRING;
    JOHN,SUSAN    :INTEGER;

JOHN:=5;

NAME:='JOHN'  (*THIS LINE IS QUITE DIFFERENT FROM*)
              (*THE FOLLOWING ONE*)
SUSAN:=JOHN;
```

Standard PASCAL does not have the STRING data type. Character strings are simulated within a subrange of data types known as an array. You will be introduced to arrays later in the course.

Assignment 5

Flowchart and code the following problem:

The local weather bureau needs a computer program to aid with its monthly statistics. At the end of each month the bureau would like to input into the computer daily its morning, afternoon, and evening temperature readings and display the average daily temperature.

In addition, after all the days of the month have been processed, the average morning, afternoon, and evening temperatures for the month will be displayed.

Your program should prompt appropriately for each set of input. The output should be labelled clearly.

Since any given month may have 31, 30, or 28 days (your program need not be set up to handle a leap year), the program should initially read in a numerical value between 1 and 12 representing the month of the year in order to determine the correct amount of input data to follow. Your program should be able to translate the numerical representation of the month to the character representation shown in the sample output.

SAMPLE INPUT:

```
2
30 32 34
28 28.5 31.4
   .
   .
   .
```

SAMPLE OUTPUT:

```
AVERAGE TEMPERATURE FOR FEBRUARY 1 WAS 28 DEGREES.
AVERAGE TEMPERATURE FOR FEBRUARY 2 WAS 29.3 DEGREES.
   .
   .
   .

AVERAGE MORNING TEMPERATURE:    28 DEGREES
AVERAGE AFTERNOON TEMPERATURE: 30 DEGREES
AVERAGE EVENING TEMPERATURE:    31.5 DEGREES
```

Be sure to draw a storage layout with your flowchart (details at the beginning of this chapter). The starting value of any initialized variable should be shown in the storage layout. (Put the correct value inside the box.) Notice there are three distinct timings for this program, the "Initialization" (or beginning), the "main loop" and the "ending" where the final averages are printed out.

NOTE: FOR THIS ASSIGNMENT AND ALL FUTURE ASSIGNMENTS WHERE SAMPLE INPUT AND SAMPLE OUTPUT DATA IS PROVIDED, YOUR INPUT AND OUTPUT WILL NOT APPEAR ON YOUR TERMINAL IN THE SAME FORMAT THAT IT IS PRESENTED IN THE STUDY GUIDE UNLESS YOU:

a. read you input from the KEYBOARD (consult the APPLE-PASCAL Language Reference Manual)

b. read the input from a disk file (Disk files may be covered in this course at the discretion of the instructor.).

Otherwise your program input on output will be displayed on alternate lines. Consult your instructor regarding the format required for each assignment.

Assignment 6

Character data play an important part in many computer problems. Notice that even number symbols are part of the character set. If numbers are not participating in any arithmetic operations it is often better to describe them in characters.

Referring to the flowchart and storage layout at the end of this chapter, code and run the following problem.

Write a program that reads in four words at a time, each word being no longer than 10 characters; the program is to print out each set of words on a separate line, in alphabetical order. After the last set of words has been processed the program is to print out:

a. the number of sets processed

b. the number of sets that were in alphabetical order to start with, and needed no rearranging.

Use the value "STOP" for WORD1 to signal the end of data. The line containing this value should not be treated as valid input data.

Flowchart Notes

The basic approach taken here is to:

1. get the "largest" word into WORD4 position. (The first three decision steps do this.)

2. get the "second largest" word into WORD3 position. (The fourth and fifth decision steps do this.)

3. get the first and second words into their proper positions. (The sixth decision step does this.)

4. "Swapping" is accomplished by three steps

   a. Move "high value word" to HOLD.

   b. Move "low value word" to "low value box".

   c. Move "HOLD" word to "high value box".

5. The SWITCH variable is used to record the occurrence of a "swap". Since any "swap" means rearranging was needed, the IN-ORDER counter is not increased by 1 when SWITCH is still = 'TRUE'.

Input Considerations

If you use STRING variables, e.g., STRING[10], you will have to type in one variable per line on the input screen, i.e., your program will need four READLN statements to read in four words.

This is not inconsistent with the flowchart symbol implying that the four words are to be read in a single statement. The flowchart shows that at this point of the algorithm four words must be read into the program. Even though the reading of these words is a little awkward with APPLE-PASCAL, the algorithm is not changed.

You should also use READLN and not READ for inputting STRING variables.

You might also consider using a PACKED ARRAY OF CHAR. This method would allow you to read in four words at a time from a single input line. You would need to arrange your words in 'fields', i.e., for PACKED ARRAY[1..10] OF CHAR variables, 10 columns must be used for each word in the input line (Unused columns should be filled with blanks.).

Note however, that APPLE-PASCAL has difficulty in dealing with PACKED ARRAYs of CHAR. PACKED ARRAYs cannot be read in and arrays that are not packed cannot be manipulated within the code as single unit character strings.

Use the following input data to test your program:

```
SALESMAN  MONTH  PROGRAM  WRITE
SEE  THE  BLUE  SKY
CAT  THROUGH  A  RAN
THE  TOP  TUMBLED  TO
WHENEVER  WATER  WAS  WHITE
A  BLUE  CLEAR  SKY
STOP  ANY  ANY  ANY
```

Sample Output:

Check your answers against the following before handing in your assignment.

MONTH	PROGRAM	SALESMAN	WRITE
BLUE	SEE	SKY	THE
A	CAT	RAN	THROUGH
THE	TO	TOP	TUMBLED
WAS	WATER	WHENEVER	WHITE
A	BLUE	CLEAR	SKY

NUMBER OF CARDS = 6 NUMBER IN ORDER = 1

Don't forget to hand in your assignment with the full documentation requirements as specified in the study guide, appendix B.

A flowchart need not be handed in if your program follows the one given in the study guide exactly. Otherwise hand in a flowchart corresponding to the code you are handing in.

Solution 1:

```
WHILE NUMBER > -99 DO
    BEGIN
        READLN (NUMBER);
        WRITELN (NUMBER)
    END;
```

Solution 2:

```
READLN (NUMBER);
WHILE NUMBER > -99 DO
    BEGIN
        WRITELN (NUMBER);
        READLN (NUMBER)
    END;
```

FIGURE IV.A

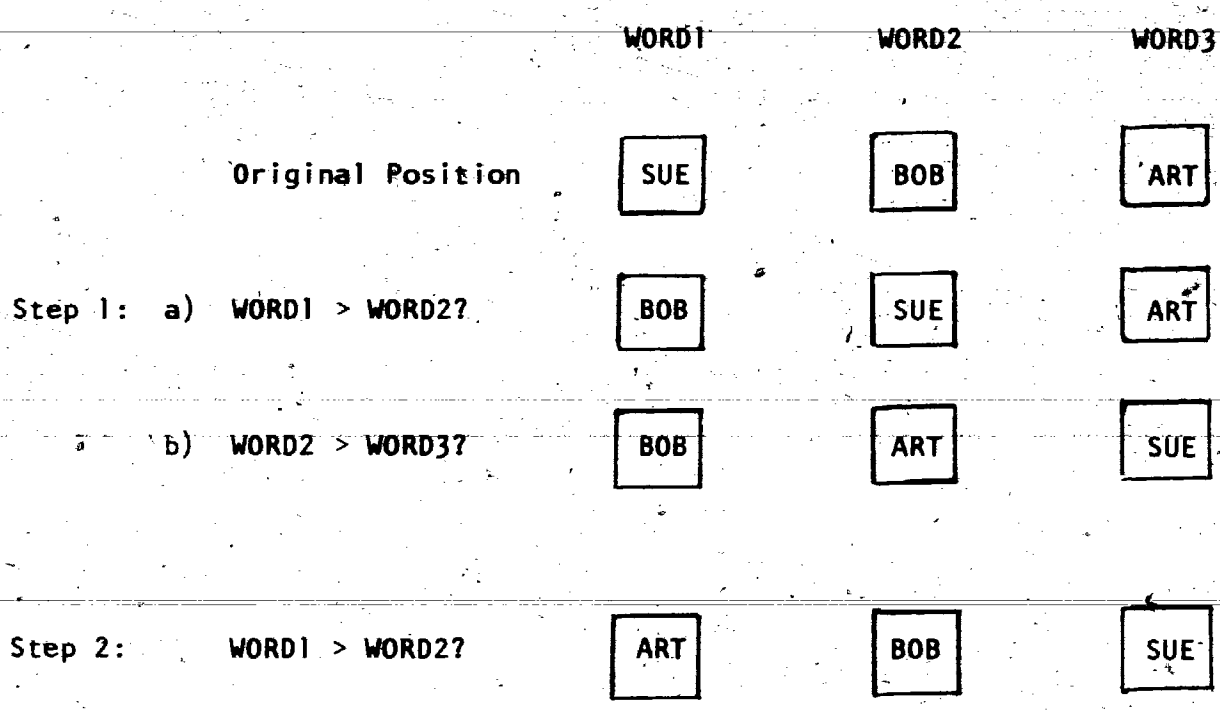

WORD1
WORD2
WORD3
Original Position
SUE
BOB
ART
Step 1: a) WORD1 > WORD2?
BOB
SUE
ART
b) WORD2 > WORD3?
BOB
ART
SUE
Step 2: WORD1 > WORD2?
ART
BOB
SUE

FIGURE IV.B

Algorithm to Exchange the Contents of Two Variables

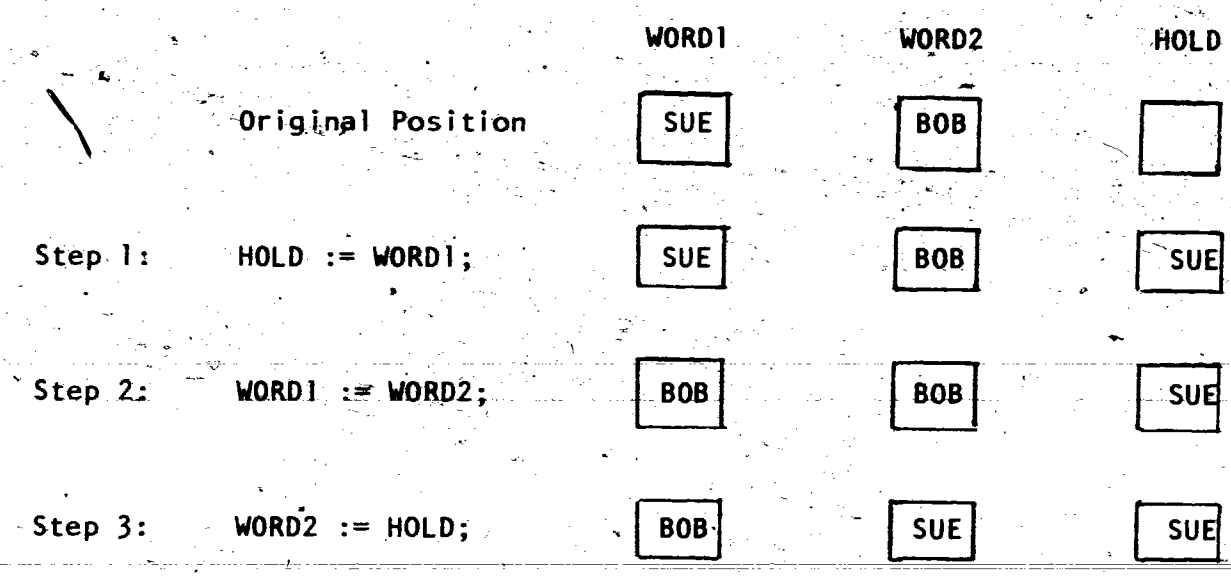


FIGURE IV.C

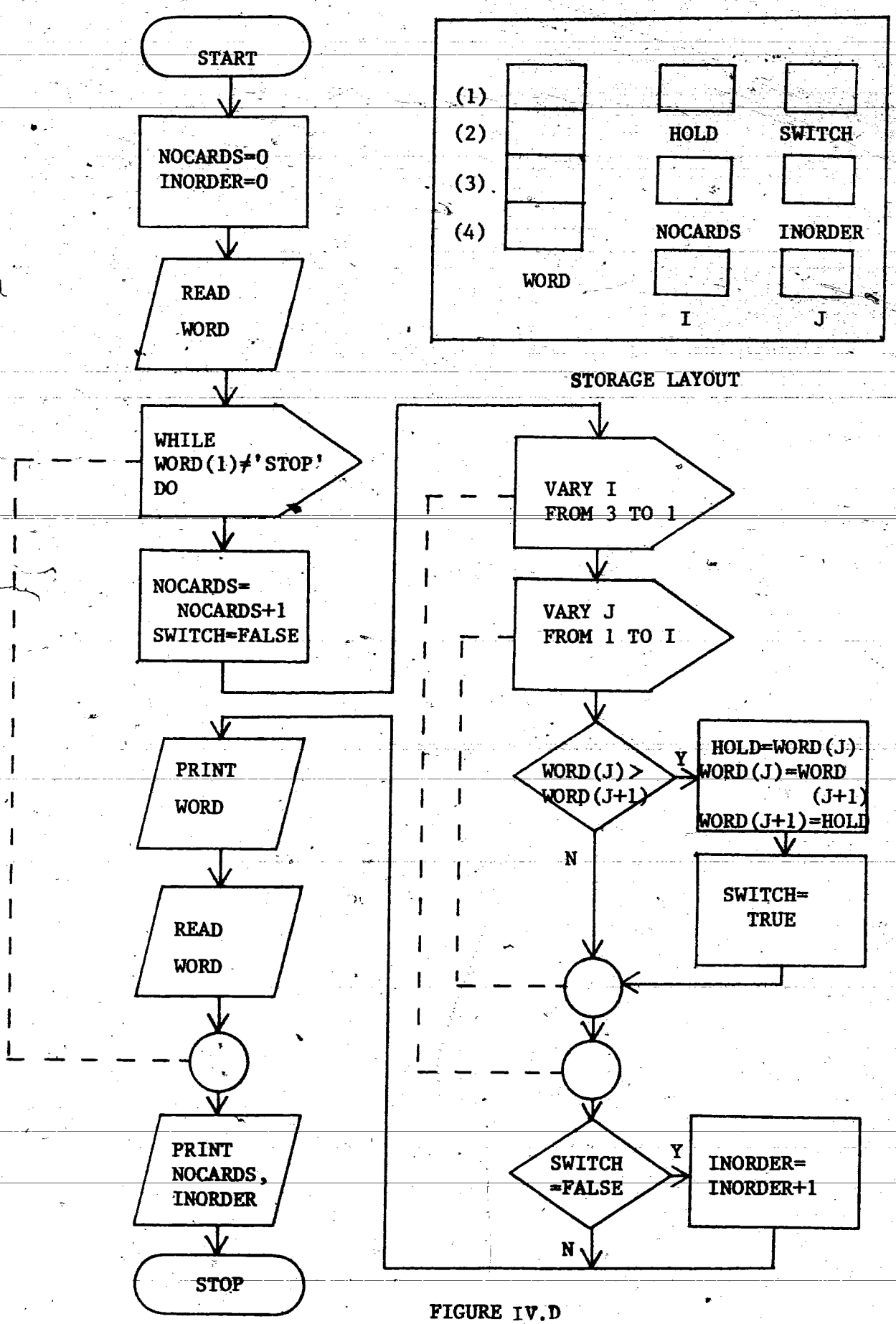

START
NOCARDS=0
INORDER=0
READ
WORD
WHILE
WORD(1)≠'STOP'
DO
NOCARDS=
NOCARDS+1
SWITCH=FALSE
PRINT
WORD
READ
WORD
PRINT
NOCARDS,
INORDER
STOP
(1)
(2)
(3)
(4)
WORD
HOLD
SWITCH
NOCARDS
INORDER
I
J
STORAGE LAYOUT
VARY I
FROM 3 TO 1
VARY J
FROM 1 TO I
WORD(J) >
WORD(J+1)
Y
N
HOLD=WORD(J)
WORD(J)=WORD
(J+1)
WORD(J+1)=HOLD
SWITCH=
TRUE
SWITCH
=FALSE
Y
N
INORDER=
INORDER+1

FIGURE IV.D

CHAPTER V

When solving a "bigger" problem it is often desirable to think of the program as a number of procedures (parts). Each procedure may be coded and tested separately and then gradually brought together to form one program. This approach is often referred to as "modular" programming.

You will find as your programs become larger, you will have to make better use of the storage available.
Rather than coding a similar procedure several times over, each time with a different set of variables, we can use subroutines and functions to solve the problem in a more efficient manner.

Part A:

Read chapter 7 of "Introduction to PASCAL Including UCSD PASCAL".

One of the most important concepts of subprograms (procedures - i.e., subroutines and functions) is the relationship between "actual parameters" and "formal parameters".

By the use of "actual parameters" and "formal parameters", a subroutine or function can perform the same sequence of operations repetitively for the many different sets of data sent to it.

The technique of using a single piece of code to perform the same operations on several sets of data is a very important concept to master. For example:

Let's take a look at a sorting problem similar to that of Assignment 6. In this problem we will sort four numbers into ascending order. You will recall from Assignment 6, that when sorting four words into ascending order, the three "swap" steps,

a. Move "high value word" to HOLD,

b. Move "low value word" to "low value position" and

c. Move "HOLD" word to "high value position"

may need to be performed a maximum number of six times. Sorting ten pieces of data into ascending order could require the above sequence to be performed fourty-five times.

The general method of overcoming this problem is to code a subroutine to do the work of "swapping". We send the two variables at a time as "actual parameters" to the subroutine's "formal parameters".

We now call the subroutine for as many comparisons as are needed to complete the sorting process. Each time the subroutine is called, two new variables are supplied as "actual parameters" to the "formal parameters" of the subroutine.

Note that this method requires that "parameters" are "passed (or called) by reference".

Make sure that you know the difference between "call by value" and "call by reference" as described in the textbook and summarized on page 110.

The above example is illustrated on the next two pages using the numbers 9, 7, 5, and 3 as input data. Note how the sorting process is accomplished. You should be able to follow the changes in the order of the data after each call is made to the procedure COMPARE.

If you cannot follow the logic of this program, come and ask about it at the lab.

```
1.  PROGRAM PROCTEST (INPUT,OUTPUT);
2.
3.  (*THIS PROGRAM USES A SUBROUTINE TO SORT A GROUP OF FOUR*)
4.  (*NUMBERS INTO ASCENDING ORDER.                          *)
5.
6.  VAR NUMBER1,NUMBER2,NUMBER3,NUMBER4: INTEGER;
7.
8.  PROCEDURE COMPARE(VAR COMP1,COMP2: INTEGER);
9.
10. (*THE SUBROUTINE COMPARE ARRANGES TWO NUMBERS (ONLY) AT*)
11. (*A TIME INTO ASCENDING ORDER.                          *)
12.
13.
14. VAR HOLD: INTEGER;
15. BEGIN
16.     IF COMP1 > COMP2 THEN
17.        BEGIN
18.            HOLD:=COMP1;
19.            COMP1:=COMP2;
20.            COMP2:=HOLD
21.        END;
22.
23. (*PRINT OUT THE INTERMEDIATE RESULTS AFTER EACH COMPARISON*)
24.
25.     WRITELN('INTERMEDIATE RESULT:',NUMBER1,NUMBER2,NUMBER3
26.          NUMBER4);
27. END; (*COMPARE*)
28.
29. BEGIN
30.     READLN(NUMBER1,NUMBER2,NUMBER3,NUMBER4);
31.     WRITELN('ORIGINAL ORDER:     ',NUMBER1,NUMBER2,NUMBER3,
32.          NUMBER4);
33.     WRITELN;
34.
35.     (*PUT THE HIGHEST VALUE IN NUMBER4*)
36.
37.     COMPARE(NUMBER1,NUMBER2);
38.     COMPARE(NUMBER2,NUMBER3);
39.     COMPARE(NUMBER3,NUMBER4);
40.
41.     (*PUT THE THIRD HIGHEST VALUE IN NUMBER3*)
42.
43.     COMPARE(NUMBER1,NUMBER2);
44.     COMPARE(NUMBER2,NUMBER3);
45.
46.     (*PUT THE SECOND HIGHEST VALUE IN NUMBER2*)
47.
48.     COMPARE(NUMBER1,NUMBER2);
49.
50.     WRITELN;
51.     WRITELN('ASCENDING ORDER:    ',NUMBER1,NUMBER2,NUMBER3,
52.          NUMBER4);
53. END.
```

Figure V.A

With input values of 9, 7, 5 and 3, the output is as follows:

ORIGINAL ORDER:	9	7	5	3
INTERMEDIATE RESULT:	7	9	5	3
INTERMEDIATE RESULT:	7	5	9	3
INTERMEDIATE RESULT:	7	5	3	9
INTERMEDIATE RESULT:	5	7	3	9
INTERMEDIATE RESULT:	5	3	7	9
INTERMEDIATE RESULT:	3	5	7	9
ASCENDING ORDER:	3	5	7	9

Global Variables

Global variables are those used in a subroutine or a function that belong to (were declared in) the main program, or an other procedure but were not passed to the receiving subroutine or function by way of parameters.

***** NOTE:
Avoid the use of "global variables" whenever possible. "Global variables" used in a subroutine or a function detract from the idea of good modularity and thus make a program much more difficult to write and maintain.

Part B:

Review the exercises for chapter 7.
Try as many as are necessary for you to grasp the concepts presented in this chapter.

Part C: Reflect section

1. How does a subroutine differ from a function?

2. What does the BEGIN statement do?

3. Must both the BEGIN and END statements be coded?

If you cannot answer the above questions you should re-read the material for this chapter and seek help from a T.A. or Instructor.

Flowcharts and Subprograms

Each processing module should have its own flowchart and be invoked as shown in the flowcharting section in chapter I of the study guide.

Note:

The subroutine approach to flowcharting may be used even if the code is actually written in line. A good flowchart should have a "Mainline page" that will fit on a 8-1/2" x 11" sheet using a top-down structured approach in program design.
Modular constructs are essential.

Part D: Assignment 7

The Master credit card company needs a system to keep track of its outstanding customer loans. Flowchart and code a program that will for each customer:

a. read in the customer name, the amount owed and the interest rate

b. calculate the new amount owed based on the amount currently owed + interest

c. print the results.

After all the input data have been processed the program will print out the number of outstanding customer loans and the total amount owed.

Use a value of -99 for "amount owed" to signal the end of the input data. These data should not be processed as customer data.

Use a FUNCTION for part b. and SUBROUTINES for parts a. and c.

Your main program must not perform any calculations. Its sole function must be that of a control module for the subprograms.

See the Sample I/O (Input and Output) on the next page.

Sample Input:

```
A.J.SMITH          236.09          0.06
H.HALL           25000.00          0.05
J.PRENTICE       32768.00          0.142
```

Sample Output:

```
NEW AMOUNT OWING FOR A.J.SMITH IS $250.26
NEW AMOUNT OWING FOR H.HALL IS $26250.00
NEW AMOUNT OWING FOR J.PRENTICE IS $37421.06
          .
          .
          .
```

Don't forget to hand in your assignment with the full documentation requirements as specified in appendix B of the study guide.

CHAPTER VI

Arrays and Tables

In chapter VI we want to take a look at "groups" of data which are usually referred to as "arrays" in the computer world. Some of you have already worked with the "group" concept outside the sphere of Computing Science either in the form of a "table" or a "matrix".

Often when the logic is repetitive, a "group" approach will eliminate the need for a repetitive code.

Again consider the problem of sorting sets of four words into alphabetical order and counting the number of sets processed and the number of sets that already were in alphabetical order to begin with. For the sake of simplicity, let us assume that each set contains four words.

One method we could use is to assign each word to a separate variable, i.e., WORD1, WORD2, etc., as was done in Assignment 6. This method is extremely primitive (Go back to Figure IV.D in the previous chapter of this study guide.)

However, let's review the basic approach:

1. get the "largest" word into WORD4 position. (The first three decision steps do this).

2. get the "second largest" word into WORD3 position. (The fourth and fifth decision steps do this).

3. get the first and second words into their proper positions. (The sixth decision step does this).

4. "Swapping" is accomplished by three steps

   a. Move "high value word" to HOLD.

   b. Move "low value word" to "low value box".

   c. Move "HOLD" word to "high value box".

What if the number of words to be sorted was 4000 instead of just 4. Can you figure out how many lines of code would be needed?

Using a group approach, all the words would be considered as having the same variable name WORD. Each word would be distinguished from every other word by a subscript (trailer to the name).

The subscript would always be a number WORD(1) or a variable WORD(J), e.g. (J) would always contain a number between one and the maximum number of words in the set. By varying the value of the subscript (J) we would actually be varying the WORD to be processed. Each new value of (J) would select a new WORD.

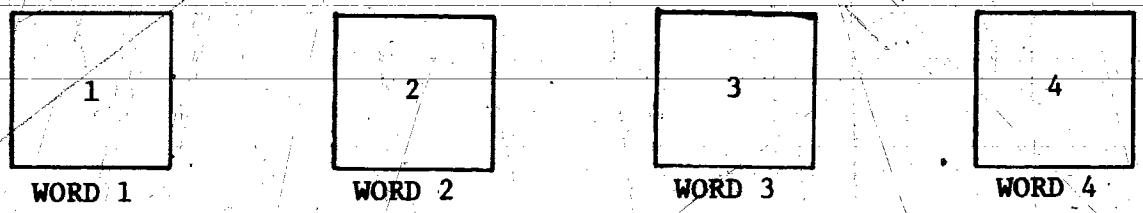
On the next two pages are flowcharts and storage layouts, one of which is the original one used for Assignment 6 and the other is a new and better approach to handling the same problem. The new solution is based on the "group" data approach. Notice, the principle is still the same, we want to put the "largest" word in the last position, WORD (4). The next "largest" word is to be put in WORD (3), the third largest in WORD (2) and the smallest in WORD (1).

The variable name (I) helps to keep track of the WORD you are trying to position. It is always one less than the actual position we are trying to finalize. This is due to the nature of the "swap" algorithm implemented inside the J loop in which the value in the J'th position is compared to the value in the J+1'th position. The variable (J) allows us to move from one word to the next, comparing and "swapping" where necessary. This "comparing and swapping" routine is executed many times.

With a set of four words I is set to 3 and the loop starts. The "comparing and swapping" routine will be processed three times for J = 1, J = 2 and J = 3. The "cycle" then repeats itself once again for I = 2, J = 1 and J = 2. The final repeat "cycle" takes place for I = 1 and J = 1. Once both inner loops have been executed for every possible value, processing continues along the main line, with the "is the current word > the next word?" decision step.

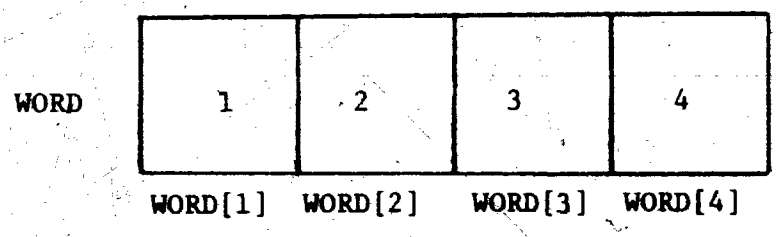
As you may have already guessed, this method of sorting is known as a "bubble sort"

The algorithms illustrated on the next pages could be made more efficient by adding an extra decision to reduce the number of steps needed to complete the sort. The implementation of the more efficient method is left for you as an exercise.




```
WORD 1:= 1;
WORD 2:= 2;
WORD 3:= 3;
WORD 4:= 4;
```

FIGURE VI.A



METHOD 1

```
WORD[1]:= 1;
WORD[2]:= 2;
WORD[3]:= 3;
WORD[4]:= 4;
```

METHOD 2

```
FOR I:= 1 TO 4  DO
    WORD[I]:= I
```

FIGURE VI.B

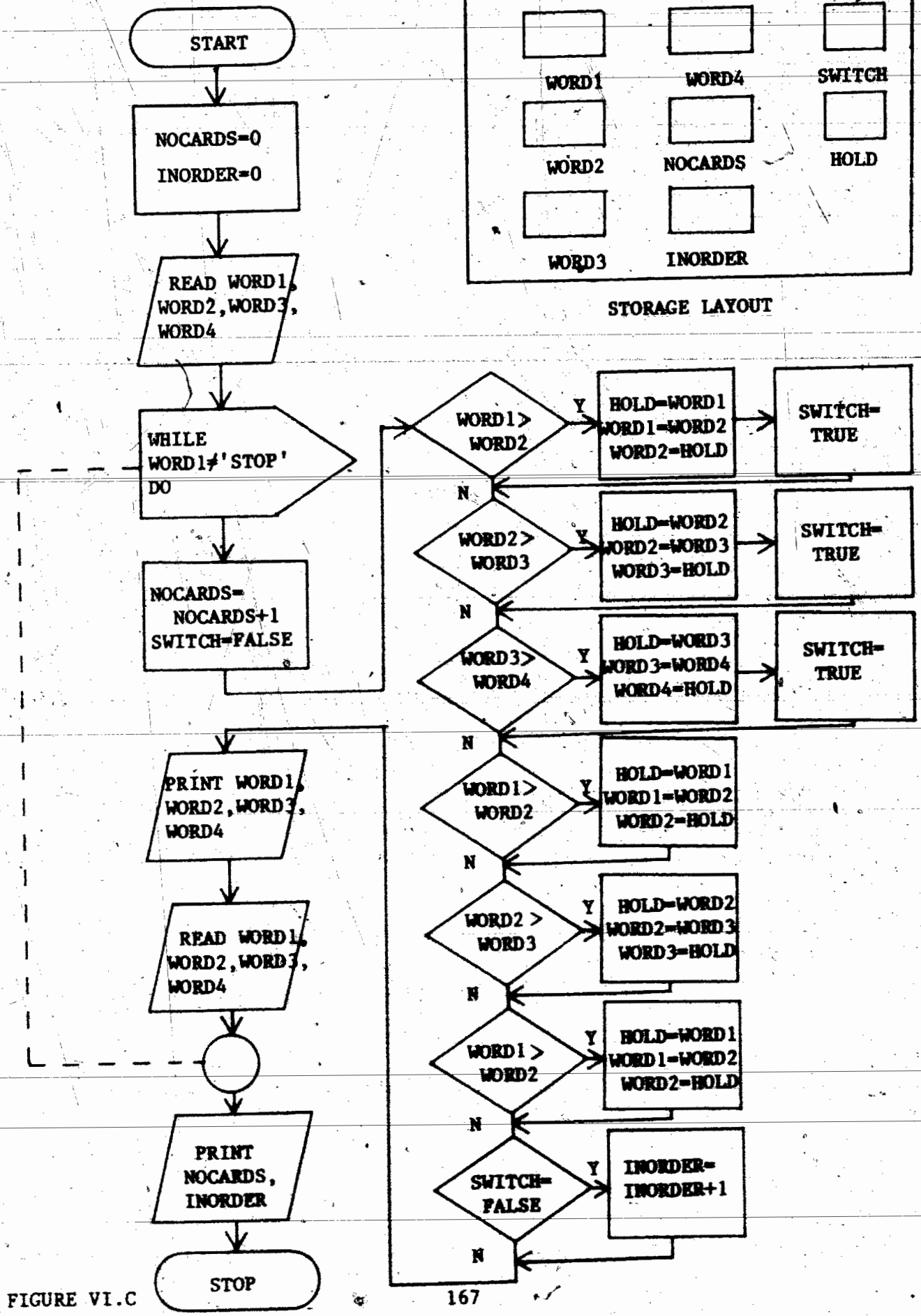

START
NOCARDS=0
INORDER=0
READ WORD1, WORD2, WORD3, WORD4
WHILE WORD1≠'STOP' DO
NOCARDS= NOCARDS+1
SWITCH=FALSE
WORD1> WORD2
HOLD=WORD1
WORD1=WORD2
WORD2=HOLD
SWITCH= TRUE
WORD2> WORD3
HOLD=WORD2
WORD2=WORD3
WORD3=HOLD
SWITCH= TRUE
WORD3> WORD4
HOLD=WORD3
WORD3=WORD4
WORD4=HOLD
SWITCH= TRUE
WORD1> WORD2
HOLD=WORD1
WORD1=WORD2
WORD2=HOLD
WORD2> WORD3
HOLD=WORD2
WORD2=WORD3
WORD3=HOLD
WORD1> WORD2
HOLD=WORD1
WORD1=WORD2
WORD2=HOLD
SWITCH= FALSE
INORDER= INORDER+1
Y
N
PRINT WORD1, WORD2, WORD3, WORD4
READ WORD1, WORD2, WORD3, WORD4
PRINT NOCARDS, INORDER
STOP
WORD1
WORD4
SWITCH
WORD2
NOCARDS
HOLD
WORD3
INORDER
STORAGE LAYOUT

FIGURE VI.C

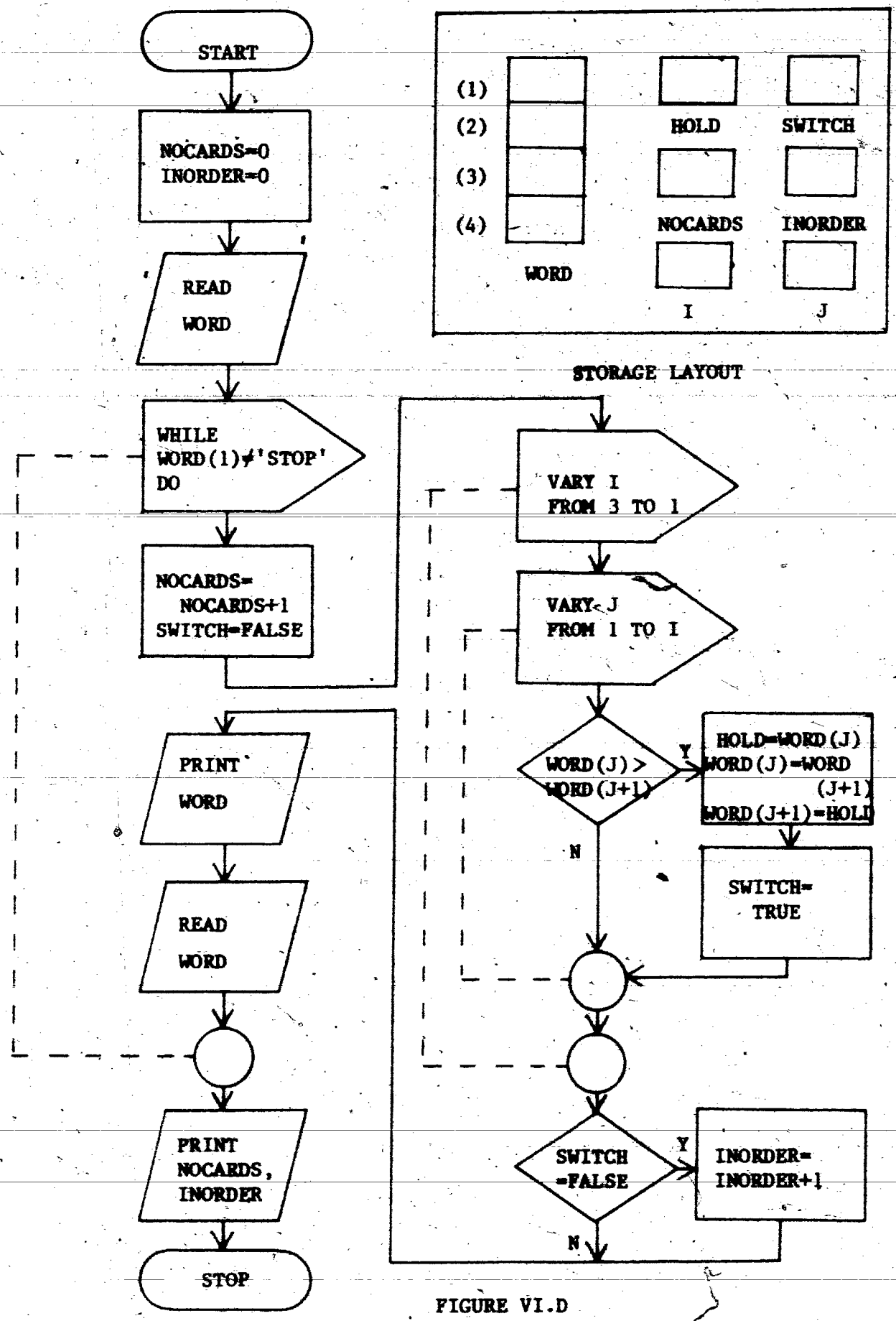

START
NOCARDS=0
INORDER=0
READ
WORD
WHILE
WORD(1)≠'STOP'
DO
NOCARDS=
NOCARDS+1
SWITCH=FALSE
PRINT
WORD
READ
WORD
PRINT
NOCARDS,
INORDER
STOP
(1)
(2)
(3)
(4)
WORD
HOLD
SWITCH
NOCARDS
INORDER
I
J
STORAGE LAYOUT
VARY I
FROM 3 TO 1
VARY J
FROM 1 TO I
WORD(J) >
WORD(J+1)
Y
N
HOLD=WORD(J)
WORD(J)=WORD
(J+1)
WORD(J+1)=HOLD
SWITCH=
TRUE
SWITCH
=FALSE
Y
N
INORDER=
INORDER+1

FIGURE VI.D

```
1.      FOR I:=3 DOWNTO 1 DO
2.         FOR J:=1 TO I DO
3.            IF WORD[J] > WORD[J+1] THEN
4.               BEGIN
5.                  HOLD:= WORD[J];
6.                  WORD[J]= WORD[J+1];
7.                  WORD[J+1]:= HOLD;
8.                  SWITCH:= TRUE
9.               END;
```

I loop	J loop	IF WORD[J] > WORD[J+1] decision
3	1	WORD[1] > WORD[2]
	2	WORD[2] > WORD[3]
	3	WORD[3] > WORD[4]
2	1	WORD[1] > WORD[2]
	2	WORD[2] > WORD[3]
1	1	WORD[1] > WORD[2]

FIGURE VI.E

User-Defined Data types

When working with a high level language, it is important for the code to be self-documenting, that is, a programmer should be able to tell what a certain part of a program does just by reading the code.

There does not yet exist a high level programming language that allows this goal to be realized fully. The PASCAL language tries to help in this area by allowing the programmer to declare his own user-defined data types to suit the type of problem he is working on. For example, a payroll program which reads in an employee's "hours worked" and "rate per hour" for each day of the week might use a FOR loop such as this:

```
FOR I:= 1 TO 5 DO
    READ (HOURS[I],RATE[I]);
```

where I is declared as INTEGER, and HOURS and RATE are REAL arrays.

On the other hand, a declaration at the beginning of the program of:

```
TYPE WORKDAYS = (MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY);
VAR DAY :WORKDAYS;
    HOURS,RATE :ARRAY[WORKDAYS] OF REAL;
```

would allow the following more descriptive loop to be used:

```
FOR DAY:= MONDAY TO FRIDAY DO
    READ(HOURS[DAY],RATE[DAY]);
```

The ability to identify data by means other than just plain numbers would be especially helpful when dealing with data involving I/O (input/output) operations. Unfortunately most versions of PASCAL including the one we are using do not allow user-defined data types to be read in or printed out. This severe restriction severely limits the usefulness of this excellent feature.

Part A:

Now let's see what the "Introduction to PASCAL Including UCSD PASCAL" has to say about data types and arrays. Read chapters 8 and 9 of "Introduction to PASCAL Including UCSD PASCAL".
Pay special attention to the UCSD features presented at the end of chapter 9. It is in the use of arrays and STRING variables that you will find the greatest difference between UCSD PASCAL and Standard PASCAL. The UCSD language extentions are the more advantageous.

Note the error on page 144 (middle of the page); the equal sign should be a colon, i.e.,

```
VAR POSITION = ARRAY [1..26] OF CHAR;
```

should be

```
VAR POSITION : ARRAY [1..26] OF CHAR;
```

You should by now be quite familiar with the APPLE-PASCAL Operating System Reference Manual as well as the APPLE-PASCAL Language Reference Manual. There are many helpful points in these books that you can pick up in a lecture or from a study guide.

You must learn to assimilate the information presented in the manuals into your programming experience in order to enhance your understanding of programming and the computer system you are working with.

Part B:

Don't forget to try the exercises in the text.

Part C: Reflect section

See if you can write PASCAL code to produce the following output;

```
X[1,1]=0    X[1,2]=12   X[1,3]=13   X[1,4]=14   X[1,5]=15
X[2,1]=21   X[2,2]=0    X[2,3]=23   X[2,4]=24   X[2,5]=25
X[3,1]=31   X[3,2]=32   X[3,3]=0    X[3,4]=34   X[3,5]=35
   .           .           .           .           .
   .           .           .           .           .
   .           .           .           .           .
X[25,1]=251 X[25,2]=252 X[25,3]=253 X[25,4]=254 X[25,5]=255
```

Note:

There is no input data for this problem. Answers not understood fully should be questioned.

Part D: Assignment 8

Shortly before Mother's Day, the Lactose Greeting Card Company prepares a supply of 10,000 cards for each one of its 20 different types. Orders for these cards come in on data cards; each card has three fields: order number, card and quantity. Write a program that processes these orders by changing the supply available for that card type and printing an output line showing the order number, number of cards ordered, number of cards sent, card type and supply still available. Do not print a value for supply, if the supply for that card type is zero. (See sample output). If an order cannot be filled because the requested quantity exceeds the supply, send whatever is available. If nothing is available "send" zero.

Processing ends when an order number of -1 is read into the program.

When all the orders have been processed, the program should print out in descending order, the amount of stock remaining for each type of card whose supply is less than 9000. (A remaining supply of zero is also printed.)

Sample Input:

Order number,	Type,	Quantity
375	7	3750
41502	15	2500
11402	21	1500
32801	12	4000
11512	15	5000
40900	17	1000
15113	15	2500
700	7	9025
40906	17	3378
61521	7	750
-1	1	1

Sample Output:

LACTOSE GREETING CARD COMPANY
SALES LIST FOR JANUARY

ORDER NO.	QUANTITY	SENT	TYPE	SUPPLY
375	3750	3750	7	6250
41502	2500	2500	15	7500
ORDER NUMBER:		11402	INCORRECT TYPE	
32801	4000	4000	12	6000
I1512	5000	5000	15	2500
40900	1000	1000	17	000
15113	2500	2500	15	
700	9025	6250	7	
40906	3378	3378	17	5622
61521	750	0	7	

CARD TYPES WITH A REMAINING SUPPLY OF LESS THAN 9000.

TYPE	SUPPLY
12	6000
17	5622
15	0
7	0

Flowchart and code the above problem. To make best use of your time do the following:

1. Set up a storage layout for the required data.

2. Develop the flowchart by putting down what you would have to do to process the sample input without the use of a computer.

3. If you are not sure your logic is correct, check your flowchart with an instructor or T.A.

4. Review any PASCAL statements necessary for coding.

5. Code your program.

6. Desk check your program for clerical errors (spelling, use of comments, parentheses, etc.)

7. Desk check your program for logic errors by "playing computer" using the sample input data. Follow each step through changing the storage layout when required.

8. Check your output. Make sure it is the same as the sample output.

9. If you plan your program efficiently, the main body, excluding the sorting step and error checks, should not require more than 1 decision.

Don't forget to hand in your assignment with the full documentation requirements as specified in appendix A of the study guide.

CHAPTER VII

In this chapter we want to look at an other way of grouping data called RECORDS. We already know generally, that the best way to handle a group of items is to use an array. However what if we have a group of items involving different types of data, i.e., a list of club members, we would still want to keep member information such as name, address, phone number and dues paid grouped together for each member.

Name, address and phone number can be handled as STRING variables (or PACKED ARRAY OF CHAR). However, dues paid must be stored as INTEGER or REAL since addition and subtraction are normally performed with this information. Since an array in PASCAL can be of only a single data type, we must look to an other type of structure called a RECORD to assist us with our grouping.

How should a RECORD be arranged? There is no one single way to arrange a RECORD. This question can be best answered by the programmer and the final user of the program. The main point to remember is that RECORDS exist in PASCAL to help the programmer organize the program in a logical fashion for humans.

Part A:

Read chapter 10 of "Introduction to PASCAL Including UCSD PASCAL". Pay special attention to the "Case Study 1" and "Case Study 2" as well the section on "Variants" (pages 203-205).

***** Note the error on page 190:

WITH EMPLOYEERECORD DO

should be

WITH FULLTIMER DO

EMPLOYEERECORD is a RECORD data type. It is a template for records that can now be created with the VAR declaration. As a data type, EMPLOYEERECORD does not have reserved storage locations and so it cannot hold information. In other words it is a model. Using this model, the programmer may create records with storage locations to hold actual data.

Part B:

Even though you may not do all the exercises, you should study all the answers at the back of the textbook.
All of the exercises are strongly recommended as practice items.

You should have a clear idea of the solution algorithms for the exercises by the time you complete the assignment at the end of this chapter.

Part C: Reflect Section

The WITH statement can be quite tricky. Remember that it is there to help you, the programmer, to save time. If you use it in such a way as to make your code incomprehensible, the WITH statement can be more of a hindrance than help.

Note that the WITH statement does away with the need for repetitive notation. It does not allow you to make multiple assignments to fields with field identifiers of the same name. After reviewing the program discussed on the lecture tape (Figure VII.A), consider the example in Figure VII.B. Examine the output from the four WRITELN statements (shown after the program listing).

```
 1 | PROGRAM RECORDS (INPUT,OUTPUT);
 2 |
 3 | (*THIS PROGRAM SORTS RECORDS BY FIELDS*)
 4 |
 5 | CONST NUMBER=5;
 6 | TYPE EMPLOYEE = RECORD
 7 |                  NAME     :PACKED ARRAY[1..20] OF CHAR;
 8 |                  POSITION :PACKED ARRAY[1..15] OF CHAR;
 9 |                  SALARY   :REAL;
10 |                 END;
11 |                 COMPANY = ARRAY[1..NUMBER] OF EMPLOYEE;
12 |
13 | VAR EMPLOYEES :COMPANY;
14 |     HOLD      :EMPLOYEE;
15 |     I,J       :INTEGER;
16 |
17 | PROCEDURE WRITEREC (VAR EMPLOY :COMPANY);
18 |
19 | (*THIS PROCEDURE WRITES OUT THE EMPLOYEE RECORDS*)
20 |
21 | VAR I,J          :INTEGER;
22 |
23 | BEGIN
24 |     FOR I:= 1 TO NUMBER DO
25 |         WITH EMPLOYEES[I] DO
26 |           BEGIN
27 |               FOR J:= 1 TO 20 DO WRITE(NAME[J]);
28 |               FOR J:= 1 TO 15 DO WRITE(POSITION[J]);
29 |               WRITELN(SALARY:7:2)
30 |           END;
31 |     WRITELN
32 | END; (*WRITEREC*)
33 |
34 | BEGIN
35 |     FOR I:= 1 TO NUMBER DO
36 |         WITH EMPLOYEES[I] DO
37 |             BEGIN
38 |                 FOR J:= 1 TO 20 DO READ(NAME[J]);
39 |                 FOR J:= 1 TO 15 DO READ(POSITION[J]);
40 |                 READLN(SALARY)
41 |             END;
42 |
43 |     (*ECHO THE INPUT DATA AS IT IS READ IN*)
44 |     WRITELN('              ORIGINAL ORDER'); WRITELN;
45 |     WRITEREC(EMPLOYEES);
```

Figure VII.A (continued on next page)

```
46
47      (*SORT RECORDS INTO ALPHABETICAL ORDER BY NAME*)
48      (*AND PRINT THEM OUT                          *)
49
50      FOR I:= NUMBER-1 DOWNTO 1 DO
51        FOR J:= 1 TO I DO
52          IF EMPLOYEES[J].NAME > EMPLOYEES[J+1].NAME THEN
53              BEGIN
54                  HOLD:=EMPLOYEES[J];
55                  EMPLOYEES[J]:=EMPLOYEES[J+1];
56                  EMPLOYEES[J+1]:=HOLD
57              END;
58      WRITELN('        ALPHABETICAL ORDER BY NAME'); WRITELN;
59      WRITEREC(EMPLOYEES);
60
61      (*SORT RECORDS INTO ASCENDING ORDER BY SALARY*)
62      (*AND PRINT THEM OUT                         *)
63
64      FOR I:= NUMBER-1 DOWNTO 1 DO
65       FOR J:= 1 TO I DO
66        IF EMPLOYEES[J].SALARY > EMPLOYEES[J+1].SALARY THEN
67            BEGIN
68                HOLD:=EMPLOYEES[J];
69                EMPLOYEES[J]:=EMPLOYEES[J+1];
70                EMPLOYEES[J+1]:=HOLD
71            END;
72      WRITELN('        ASCENDING ORDER BY SALARY'); WRITELN;
73      WRITEREC(EMPLOYEES)
74
75  END.
```

Figure VII.A (continued on next page)

Output:

ORIGINAL ORDER

JAMES, SUSAN	SR. PROGRAMMER	1900.00
WONG, MICHAEL	JR. PROGRAMMER	1700.00
PETERSON, PAT	ANALYST	2000.00
PIERCE, MARY	JR. OPERATOR	1000.00
MARCUS, STEVE	SR. OPERATOR	1400.00

ALPHABETICAL ORDER BY NAME

JAMES, SUSAN	SR. PROGRAMMER	1900.00
MARCUS, STEVE	SR. OPERATOR	1400.00
PETERSON, PAT	ANALYST	2000.00
PIERCE, MARY	JR. OPERATOR	1000.00
WONG, MICHAEL	JR. PROGRAMMER	1700.00

ASCENDING ORDER BY SALARY

PIERCE, MARY	JR. OPERATOR	1000.00
MARCUS, STEVE	SR. OPERATOR	1400.00
WONG, MICHAEL	JR. PROGRAMMER	1700.00
JAMES, SUSAN	SR. PROGRAMMER	1900.00
PETERSON, PAT	ANALYST	2000.00

Figure VII.A

```
PROGRAM WITHTEST (INPUT,OUTPUT);
TYPE REC1 =
    RECORD
        DAY,
        NIGHT: INTEGER
    END;

     REC2 =
    RECORD
        DAY,
        EVENING: INTEGER
    END;

     REC3 =
    RECORD
        T1: REC1;
        T2: REC2
    END;
VAR TEST: REC3;

BEGIN
    TEST.T1.DAY:=1;
    TEST.T1.NIGHT:=2;
    TEST.T2.DAY:=3;
    TEST.T2.EVENING:=4;
    WRITELN('      T1.DAY   T1.NIGHT   T2.DAY   T2.EVENING');
    WRITELN(TEST.T1.DAY,TEST.T1.NIGHT,
        TEST.T2.DAY,TEST.T2.EVENING);

    WITH TEST,T1,T2 DO
        DAY:=5;
    WRITELN(TEST.T1.DAY,TEST.T1.NIGHT,
        TEST.T2.DAY,TEST.T2.EVENING);

    WITH TEST DO
        BEGIN
           WITH T1 DO
                DAY:=6;
           WITH T2 DO
                DAY:=7
        END;
    WRITELN(TEST.T1.DAY,TEST.T1.NIGHT,
        TEST.T2.DAY,TEST.T2.EVENING);
END.
```

Output:

```
      T1.DAY   T1.NIGHT   T2.DAY   T2.EVENING
         1         2         3         4
         1         2         5         4
         6         2         7         4
```

Figure VII.B

Part D: Assignment 9

This assignment provides an excellent opportunity to make use of modular programming plus implementing the structured programming techniques learned so far. You will find the assignment easily lends itself to many "processing modules".

You are expected to effect a good top-down design as well as to employ a minimum of one subroutine and one function. "Actual parameters" and "formal parameters" MUST be used with all subroutines and functions.

Flowchart, code and run the following problem:

Every month, Modern Computing, a monthly magazine, sends out subscription notices to all customers whose subscriptions are up two months hence or have already expired. (Expired subscriptions are kept on record for a period of one year before being deleted from the active files). The notice consists of a detailed billing for the next subscription period, the length of which is determined by the subsciption currently active or recently expired.

Input data for each customer consists of:

a. subscriber's name

b. subscriber's account number

c. length of subscription (ranging from 1 to 5 years)

d. expiry date of latest subscription

e. date the account with Modern Computing was first opened.

Use a RECORD to store the above information.

Rates are $8.00, $15.00, $21.00, $26.00 and $30.00 for a 1 to 5 year subscription respectively. A 2 percent discount off the subscription rate is given for each year in excess of five years the account has been open. No customer, however, may receive more than a 20 percent discount.

For each notice sent out the program is to print a line of output containing the account number, name, length of subscription, rate per year, and total amount due.

After all customers' accounts have been processed (the number of customers varies from month to month, use a value of -1 for the subscriber's account number), the program is to print out a summary containing the number of subscription notices processed and the total amount owing in each separate subscription category.

Finally, the program will print out the total number of customers' accounts processed and the total amount of money owing for the month.

Note that all dates appear in the form (yymm), the first 2 digits for the year and the last 2 for the month.

Review the sample input and sample output. Think about what you would have to do without a computer as you develop your flowchart. Desk check your flowchart. You may even want to have it checked by an instructor or a T.A. Remember, coding poor logic is a waste of your time. Code carefully, watching for correct spelling and punctuation.

Hint:

A full year's subscription is considered to run from a designated month of one year to the corresponding month of the next year. i.e. February 1980 to January 1981 is only 11 months, but February 1980 to February 1981 is a full year. Therefore, it is not necessary in this assignment to break up dates into separate year and month categories in order to determine the amount of discount years, if any.

Sample Input:

```
7602   (this date is already two months hence)

J.A. KOGAN       621422  3  7512  6801

K.P. CRANE       623891  1  7608  7508

L.M. MCKAY       733311  1  7601  5904

J.J. SMITHERS    821462  5  7512  6512

B.C. NELSON      231224  5  7804  7004

A.K. BOLDER      621427  3  7602  7302

M.N. NEEDHAM     621448  1  7605  7405

R.M. BALMER      243786  3  7601  7001

P.M. ATKINS      318112  5  7910  6910

F.F. LO          614422  3  7601  7301

K.L. SIMMS       715598  3  7702  7402

DUMMY           -000001  1  0000  0000
```

Sample Output:

ACCT. NO.	NAME	YEARS	RATE	AMT. DUE
621422	J.A. KOGAN	3	6.72	20.16
733311	L.M. MCKAY	1	6.40	6.40
821462	J.J. SMITHERS	5	5.40	27.00
621427	A.K. BOLDER	3	7.00	21.00
243786	R.M. BALMER	3	6.86	20.58
614422	F.F. LO	3	7.00	21.00

YEARS OF SUBSCRIPTION	ACCOUNTS DUE	AMOUNT DUE
1	1	6.40
2	0	0.00
3	4	82.74
4	0	0.00
5	1	27.00

TOTAL NO. OF RENEWALS: 6

TOTAL BILLINGS: 116.14

NOTE:
The output should be in columnar form but it does not have to be exactly the same as the above.

CHAPTER VIII

In chapter 11 of "Introduction to PASCAL Including UCSD PASCAL" we will take a closer look at more sophisticated ways of treating input and output data through the use of files.

While this course does not focus on PASCAL file handling, you should at least know how to read a text file from the disk and how to write a text file to the disk. This aspect of file manipulation should be used in your last assignment and in your final project.

The practice of reading in dummy data to detect the end of input into a program is a primitive practice.
The EOF statement allows a PASCAL program to detect the end-of-file condition, that is, when the end of the input data has been reached. This feature was discussed in chapter 5 of "Introduction to PASCAL Including UCSD PASCAL". This means that the end-of-file is automatically marked for you at the end of the input data and that your program can detect this mark by using the EOF statement. Similarly, the end of a line has a end-of-line marker (EOLN).

Although up until now you have been working only with interactive programs, you should realize that the input data that entered into the program from the terminal is also part of a file. This means that as far as APPLE-PASCAL is concerned, the terminal (input and output) and the printer (output only) are files.

Part A:

Read chapter 11 of "Introduction to PASCAL Including UCSD PASCAL".
Also, (re)read the sections on files in both the APPLE-PASCAL Language Reference Manual and the APPLE-PASCAL Operating System Reference Manual.

At this stage of the course you must be able to decipher the information of the text, the language manual, and the operation manual on your own. In Assignment 10 you will be asked to type the input data into a disk file, have your program read this data from the file, and have the program print the results directly onto the printer as well as into a disk file.

Make sure you read the material selectively, taking out the information you need for your assignment.

Part B:

Take a thorough look at the exercises for chapter 11. Attempt only those which interest you and which you have time to do.

Part D: Assignment 10

The Edemdale grocery store keeps records on its produce in the following form:

Code	- character (24)
Price	- 4 digits, 2 of which are decimals
Month	- 2 digits
Day	- 2 digits

The code has several component parts. All perishable items have 'PER' appearing somewhere in the code followed by a two-character alphanumeric which indicates the acceptable shelf life for that product. For example,

DAIRY ALLPER15

indicates the item is perishable and that an "in stock period" of 15 days is normal. If the "in stock period" extends beyond this time, new discount price labels are to be created, to encourage quick sale of the article. The new labels should have the form:

```
+----------------------------------------+
|                                        |
|  CODE                      $ PRICE     |
|                                        |
+----------------------------------------+
```

Note that in the printed code, we keep only that part before the identification 'PER' (if there is any).

In order to determine the "in stock period", the date must be converted to a Julian date (total days per year count). For example,

```
JANUARY 28 = JULIAN DATE 28
APRIL 15   = JULIAN DATE (31+28+31+15)
           = 105
```

This date is then compared with today's date (which is the first entry in our data list) converted to Julian. If the difference is greater than the "in stock period", the new price is to be calculated by applying the discount rate for that "in stock period" to the original price according to the following scheme:

IN STOCK PERIOD (DAYS)	7	14	21	28	35	49	84
DISCOUNT (PERCENT)	20	18	15	12	10	8	5

Method:

a. Use a Function to determine if an "in stock period" exists. If so, then return the "in stock period", else return a zero.

b. If an "in stock period" does exist, invoke a second Function from within the first Fuction to determine what the period is.
If you use a STRING variable for the code, you can check for the occurance of a 'PER'. If a 'PER' does exist then you know that the third and fourth elements following represent the "in stock period". You now have to convert this STRING representation of the "in stock period" to a numerical form. One method to do this is as follows:

1) set up a 10-character STRING with the symbols '0' to '9' in positions 1 to 10 respectively.

2) Use a FOR loop to compare the third position after the 'PER' to each position of the 10-character STRING. When a match is found, the index of the FOR loop at that point will be the numerical value of the third element after the 'PER'.

3) Repeat the above step with the fourth element after the 'PER' to find its numerical value.

4) How to combine the numerical values derived in the above two steps to form the numerical representation of "the instock period" is left for you to complete.

Another method to accomplish this job after finding the 'PER' in the code STRING would be to use a CASE statement to represent the characters '0' to '9'.

Still another method using standard PASCAL is as follows:

```
PROGRAM PERFIND (INPUT,OUTPUT);
VAR I,J,NUMBER: INTEGER;
CODE: PACKED ARRAY[1..24] OF CHAR;

BEGIN

    I:=0;
    NUMBER:=0;

    FOR J:= 1 TO 24 DO READ (CODE[J]);
    REPEAT
       I:=I+1;
       IF CODE[I]='P' THEN
          IF CODE[I+1]='E' THEN
             IF CODE [I+2]='R' THEN
                NUMBER:=10*(ORD(CODE[I+3])-ORD('0'))+
                         (ORD(CODE[I+4])-ORD('0'))
    UNTIL (NUMBER > 0) OR (I=20);
    WRITELN(CODE,NUMBER)
END.
```

Each of the methods discussed above should be in the form of either a Procedure or a Function. If you choose to implement your subprogram from the programmed example above, you must supply very detailed documentation.

c. Code the Calculation of Julian date as a function or a procedure.

SAMPLE INPUT:

```
12 01
DAIRY A201PER14          00.35 11 30
DAIRY A24                01.29 03 20
DAIRY A275PER07          00.42 11 22
VEGETABLE B371PER14      01.09 11 16
VEGETABLE B43PER14       01.02 11 23
FRUIT K473KPER21         00.48 11 08
FRUIT K245               00.92 11 09
FRUIT K458PER28          03.26 11 03
FROZEN Z75               04.63 07 24
FROZEN Z621              05.42 03 30
FROZEN Z28PER49          02.48 10 14
FROZEN Z89PER84          04.22 09 07
DAIRY A72PER07           00.62 11 29
FLOWER W437PER07         03.22 11 23
FLOWER W24               04.62 11 26
FLOWER W68PER14          03.06 11 15
FROZEN Z54PER35          14.32 10 26
FROZEN Z24PER49          23.02 10 12
VEGETABLE B20PER35       04.28 10 23
DAIRY A114PER14          00.78 11 15
```

SAMPLE OUTPUT:

DAIRY A275	$ 0.34
VEGETABLE B371	$ 0.89
FRUIT K473	$ 0.41
FROZEN Z89	$ 4.01
FLOWER W437	$ 2.58
FLOWER W68	$ 2.51
FROZEN Z54	$12.89
FROZEN Z24	$21.18
VEGETABLE B20	$ 3.85
DAIRY A114	$ 0.64

NOTE: Output MUST include the borders.

APPENDIX A

A Guide to Writing Term Projects

Introduction

The term project should demonstrate your ability to use the computer language you have learned, as a problem solving tool.

The program should be user oriented and attempt to solve some problem in your area of interest. In most of the course assignments, you were given the input for a program, and told what the output should look like. Here, you will be responsible for defining your own program I/O requirements.

The program should contain at least 75 non-I/O statements, that is, statements not involving the PASCAL READ, PRINT or WRITE keywords. Do not "pad" your program with additional and unnecessary statments just to meet the length requirement. If you have a suitable problem, the length requirement will not be a burden.

What to do? Select a topic you are familiar with, keeping in mind the types of problems a computer can solve. It would be helpful to:

a. investigate your major field of study--talk to various faculty members.

b. consider your hobbies.

Project Proposal

A very important aspect of the project is its formal definition. This is the "Project Proposal" which MUST be handed in and APPROVED by an instructor or a T.A. before the project is begun. Some modifications may be suggested at that time.

Define the boundaries of the problem clearly so that flowchart work can be started. Your definition should resemble "User Documentation" and consist of the following:

a. User Purpose

b. Sample Input

c. Sample Output

d. A brief (10-15 line) description of how the problem will be solved.

Sample Projects

The following is a list of some of the projects which have been successfully attempted in the past:

1. Simulation of a Company. Read in cards corresponding to employees in a company, and people looking for work. Use a random number generator to hire and fire people for jobs.

2. Company Inventory. Keep track of incoming and outgoing transactions, back-orders, current stocks, etc., with monthly and yearly summaries.

3. Graphing. Draw an X-Y graph, or a bar graph for some data.

4. Advanced graphing. Draw an X-Y-Z graph, or a block graph for some data.

5. Titration Curve (chemistry). Read in pH and volume data for a titration, plot the curve, and do some calculations.

Solving the Problem

Once your project proposal has been approved, you will be ready to begin. The following steps will guide you.

a. Determine the criteria necessary to make your program work.

   1) The program may have to read in data--will you be making error checks on the user input?

   2) Will you require arrays to store data?

   3) Will you need to do operations on your data such as sorting them into a specific order?

   4) What kinds of calculations or manipulations will have to be made on the data?

b. Draw a "top-level" flowchart (Hierarchy flowchart) for your problem. Place the guidelines from (a) into logical order. At this point it would be sufficient to have a process box with "Sort the data for . . . " written in it, rather than a detailed flowchart of the actual DO-loops necessary to perform the sort.

c. Refine the steps in (b). A good approach at this stage would be to draw a small flowchart for each step.

d. Discuss your progress with an instructor or T.A. This step may save you time and grief later on.

e. Write the code for your problem. Your flowcharts from (c) should be detailed enough so that you can code directly from them. MAKE USE OF SUBROUTINES AND FUNCTIONS. A very good approach would be to make each of the "top-level" operations from (b) into a subprogram.

f. Desk-check your code. Work through it step by step, keeping track of the values of all variables. Weed out syntactic errors, such as missing commas, unmatched parentheses, etc.

g. Type your program into the APPLE-PASCAL system. If you were careful in all the preceding steps (and are a good typist), your program will work the first time.

Program Organization

You should make effective use of subroutines and functions in your program. Ideally, your main procedure should consist solely of a sequence of subprogram invocations (perhaps within a loop). Subprograms should be used when:

a. a set of operations is to be done several times. This would be particularly true if the operation is done on several different sets of data.

b. a set of operations logically belongs together. Examples might be the input section of the program, the calculation section, or the print section.

Internal Documentation

Your program should have comments in it, similar to the assignments. Every group of statements corresponding to a "top-level" operation should be preceded by a comment, but other than that, comments should be few.

What to hand in as the completed Project

The completed project will contain the following items:

a. Your project proposal showing that it has been approved by a T.A. or instructor.

b. Program, consisting of:

   1) A listing of a run of your program, operating on some input data.

   2) A COMPLETE copy of the input data used.

c. A "Top-level" flowchart (Hierarchy flowchart) ONLY!!! DO NOT hand in your detailed flowcharts.

d. User Documentation.

Parts (a) and (b) are self-explanatory.
Part (c) should be a short write-up of a maximum of three pages and oriented toward the potential user of your program. It should be neat and thorough - something that you would be proud to have duplicated and circulated.

Consider the user: - what does he need to know? What does he want from the program? Most certainly, he will not want to read large amounts of information about the coding, but will want a concise description of what goes into the program, and what comes out.

User Documentation should contain:

a. TITLE. Give your program a name representative of what it does.

b. ORIGIN. Say who wrote it, when and where.

c. PURPOSE.

d. SAMPLE INPUT.

e. SAMPLE OUTPUT.

f. SPECIAL RESTRICTIONS. Describe them briefly, for example, does your program read all 80 columns, or only 60 or 72? Will your program handle more than one set of data on a given run?

g. METHOD. Briefly mention any special algorithms used. Give formulae which are not common knowledge.

h. REFERENCES. Give the source of your algorithm or formulae. (This information is usually only necessary for science oriented programs).

REMEMBER, THERE IS NO EXTENTION TO THE FINAL PROJECT DUE DATE!

DO NOT HESITATE TO VISIT YOUR INSTRUCTOR OR T.A. IF YOU ARE HAVING TROUBLE

APPENDIX B

A Guide to Writing and Submitting Assignments

Introduction

Assignments must be submitted in the lab periods on the due dates listed in the semester handout for the course.
Put your assignment in an 8.5 X 11 Duo-Tang folder, containing the following five sections:

1. FRONT COVER LABEL - the front cover label of the folder must have the following information:

   a. your full name with the surname underlined

   b. the language you are taking

   c. the group you are in (see preregistration booklet)

2. TITLE PAGE

   a. assignment number - e.g. Assignment 8

   b. your name e.g. J.A. Smith, Student No. 95000-0000

   c. class - e.g. PASCAL Day Section, Group 1

3. USER DOCUMENTATION

   a. user purpose - what the program does, stated in every-day, non-coding language

   b. a layout showing a sample input and a sample output

4. FLOWCHART

5. PROGRAMMER DOCUMENTATION AND COMPUTER OUTPUT

a. the programmer documentation will explain everything a programmer needs to know about the program

b. the computer output should be attached by the last page only

c. DO NOT separate the pages of the printout.

Make sure you have checked the study guide for test data and modifications before beginning any assignment.

If you have any difficulty understanding any of these directions, ask the T.A. or the instructor to explain them in more detail. It will save you time and worry.

Example Assignment Writeup of Assignment 1

Appendix B contains the writeup of a completed assignment using assignment 1 as an example.
Before going on to the example, it is important that you understand the difference between "user" and "programmer" documentation.

User Documentation

The user documentation section includes the "User Purpose", "Sample Input", and "Sample Output". User documentation is designed for the benefit of the user. It is quite conceivable that this person may not have any knowledge of computing and computing programs. Therefore, a detailed explanation of what the program does, what the input should be, and what the output will look like is necessary here. Documentation should be brief and NON-TECHNICAL, yet should contain a good explanation of the program and how to use it.

Programmer Documentation

The programmer documentation is intended to describe the methods used in coding the program.
In addition to providing a short TECHNICAL explanation of what the program does, this type of documentation includes step by step explanations of each section of code, by means of program comments. If changes need be made at some time in the future, these comments are of invaluable assistance.
All programmer documentation should be short and to the point.

EXAMPLE ASSIGNMENT TITLE PAGE

CMPT 103

Assignment 1

J. A. Smith, 95000-0000

September 10, 1985

PASCAL, Day Section, Group 1

USER DOCUMENTATION

USER PURPOSE:

This program reads in two integers A and B, sums them and prints out the total.

Sample Input:

The two numbers must be integers. They are typed in at the terminal under the prompt "ENTER TWO NUMBERS TO BE ADDED" and must be separated by at least one blank space.

Sample:

```
2 3
```

Sample Output:

The output is printed on one line of the terminal.

Sample:

```
THE SUM OF   2   AND   3   IS    5
```

PROGRAMMER DOCUMENTATION

Programmer documentation consists of the comments placed between the lines of code.

They should contain any information for the programmer which may be needed to supplement what is shown in the user documentation, and in the flowchart.